NIST Special Publication 500-168

Report of the Invitational Workshop on Data Integrity

Zella G. Ruthberg
William T. Polk

Technology

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NIST

The National Institute of Standards and Technology[1] was established by an act of Congress on March 3, 1901. The Institute's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Institute conducts research to assure international competitiveness and leadership of U.S. industry, science and technology. NIST work involves development and transfer of measurements, standards and related science and technology, in support of continually improving U.S. productivity, product quality and reliability, innovation and underlying science and engineering. The Institute's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, the National Computer Systems Laboratory, and the Institute for Materials Science and Engineering.

The National Measurement

Provides the national system of ph ic Standards[2]
coordinates the system with measu liation Research
and furnishes essential services lea mical Physics
physical and chemical measureme alytical Chemistry
community, industry, and commer
services to other Government age
research; develops, produces, and
Materials; provides calibration ser
Standard Reference Data System.
following centers:

The National Engineering

Provides technology and technical nputing and Applied
sectors to address national needs a thematics
conducts research in engineering a ctronics and Electrical
efforts; builds and maintains comp gineering[2]
required to carry out this research nufacturing Engineering
neering data and measurement cap lding Technology
ment traceability services; develop Research
neering standards and code change mical Engineering[3]
engineering practices; and develop
transfer results of its research to th
consists of the following centers:

The National Computer Sys

Conducts research and provides sc rmation Systems
Federal agencies in the selection, a gineering
computer technology to improve e tems and Software
ment operations in accordance wit hnology
relevant Executive Orders, and oth nputer Security
by managing the Federal Informati tems and Network
developing Federal ADP standards hitecture
participation in ADP voluntary sta anced Systems
tific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Laboratory consists of the following divisions:

The Institute for Materials Science and Engineering

Conducts research and provides measurements, data, standards, reference materials, quantitative understanding and other technical information fundamental to the processing, structure, properties and performance of materials; addresses the scientific basis for new advanced materials technologies; plans research around cross-cutting scientific themes such as nondestructive evaluation and phase diagram development; oversees Institute-wide technical programs in nuclear reactor radiation research and nondestructive evaluation; and broadly disseminates generic technical information resulting from its programs. The Institute consists of the following divisions:

- Ceramics
- Fracture and Deformation[3]
- Polymers
- Metallurgy
- Reactor Radiation

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Gaithersburg, MD 20899.
[2]Some divisions within the center are located at Boulder, CO 80303.
[3] Located at Boulder, CO, with some elements at Gaithersburg, MD.

NIST Special Publication 500-168

Report of the Invitational Workshop on Data Integrity

Zella G. Ruthberg and William T. Polk, Editors

National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

September 1989

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Raymond G. Kammer, Acting Director

NIST

Reports on Computer Systems Technology

The National Institute of Standards and Technology (NIST) (formerly the National Bureau of Standards) has a unique responsibility for computer systems technology within the Federal government. NIST's National Computer Systems Laboratory (NCSL) develops standards and guidelines, provides technical assistance, and conducts research for computers and related telecommunications systems to achieve more effective utilization of Federal information technology resources. NCSL's responsibilities include development of technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive unclassified information processed in Federal computers. NCSL assists agencies in developing security plans and in improving computer security awareness training. This Special Publication 500 series reports NCSL research and guidelines to Federal agencies as well as to organizations in industry, government, and academia.

Library of Congress Catalog Card Number: 89-600756
National Institute of Standards and Technology Special Publication 500-168
Natl. Inst. Stand. Technol. Spec. Publ. 500-168, 377 pages (Sept. 1989)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1989

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

Report of the Invitational Workshop on Data Integrity January 25-27, 1989

Appendices

LIST OF FIGURES

ACKNOWLEDGMENTS

The editors wish to thank the many people who assisted them in the creation and assembly of this document. Karen Moran created the figures and typed extensive portions of the document. Lawrence Bassham provided tireless and invaluable assistance during the assembly of the camera ready copy, as well as proofreading many passages. The efforts of our many reviewers, especially those of Charles Dinkel and Alan Goldfine, were greatly appreciated.

EXECUTIVE SUMMARY

This report contains the proceedings of the second invitational workshop on computer integrity issues which took place on January 25-27, 1989 at NIST. The first invitational workshop took place on October 27-29, 1987 and addressed Integrity Policy in Computer Information Systems. This sudden increased interest in integrity issues was brought on by a landmark paper by David D. Clark of MIT and David R. Wilson of Ernst & Whinney [CLARK87] in April 1987. Their paper set forth the thesis that the current emphasis on the confidentiality aspect of computer security, found in the DoD criteria document [TCSEC] did not adequately address the integrity aspect of computer security and was therefore not serving the needs of the commercial sector. The model suggested by Clark/Wilson in their paper was discussed by scientists and practitioners at the first integrity workshop under the topics of Assurance, Granularity and Functions, Identity Verification, Auditability, and Correspondence of a System to Reality. These discussions generated more questions than answers and can be found in NIST Special Publication 500-160 [NIST160].

A significant other result of this first workshop was the decision by NIST to nurture the integrity policy activity by organizing and hosting one or more subsequent invitational workshops on this subject. The NIST Computer and Telecommunications Council established a Working Group, chaired by Robert H. Courtney, Jr., which devoted its time to the subject of data integrity. Its numerous discussions were instrumental in deciding on data integrity as the subject matter for this first follow-on workshop. A Planning Committee with many individuals from the Working Group (see Section 1.2 for member names), drew up an outline of the scope of this follow-on workshop. It was to discuss 1) Integrity Framework Elements (including (a) concepts of integrity, quality, and value, (b) integrity requirements, (c) quality metrics); 2) Implementation Requirements and Approaches (emphasizing functionality and properties); and 3) Implementation/Models (in relation to the Integrity Framework to be developed). (See Section 1.2 for more details).

A Call for Papers (in Appendix C) and a Strawman Paper by Robert H. Courtney, Jr. (in Appendix A) were distributed to the relevant communities and as a result eight papers were selected for presentation to stimulate discussions. These papers fit into the Planning Committee's outline and addressed such topics as data integrity models, data quality, integrity controls, and certification of transformation procedures that preserve data integrity. Invitations to the workshop were issued to persons who had a keen interest in integrity issues and were working in some aspect of it. As part of the attendee preparation for the workshop a position paper was requested on some related topic. The submitted position papers can be found in Appendix B.

The workshop took place over a period of two and one-half days with 66 attendees. With the further assistance of the Planning Committee, it was decided, prior to the workshop, that the attendees would be divided into five discussion groups and that their forcus would be 1) Operating Systems and Systems, 2) Telecommunications, 3) System Services, 4) Applications, and 5) Implementations/Models respectively. On Day-1 of the meeting, each discussion group addressed the Integrity Framework Elements in an effort to draw some consensus view on

these elements. The Strawman Paper presented a definition of data integrity and a view of data quality and data value which received mixed responses from the five discussion groups. Consequently, no consensus was reached on these aspects of the Integrity Framework Elements.

The consensus that was reached was concerned with 1) policy and objectives for data quality and 2) common mechanisms for data quality.
The policy and objectives consisted of:

- Authorization
- Accountability
- Auditability
- Internal Consistency
- Separation of Duties
- 'Real World' Correspondence
- Concepts of: Constrained Data Items
  Well-Formed Transactions
  Recoverability
- Prevention of Missing/Wrong/Extra Data
- Change Occurs If and Only If Appropriate

The common mechanisms suggested were:

- Means of Attribute Measurement
- Trusted Transformation Processes
- Constrained Data Items
- Object/Subject Authorization
- Event Signalling
- TCB
- Application Specific Reference Monitors
- Separation of Duties
- Privilege Mechanisms
- Audit Trails, Logging Mechanisms

This consensus formed the basis for the Day-2 discussions.

In the Day-2 discussions of the five groups, only Telecommunications concluded that no new technological advances were necessary to achieve data integrity objectives. The System Services group, which focused on DBMS for lack of more time, concluded that existing mechanisms in DBMSs could be used for integrity control, but technological advances are needed to make that control more effective and flexible. The Applications and Operating Systems groups concluded that technological advancement is needed in those areas to achieve satisfactory integrity control. The Implementations/Models group appears to be furthest from meeting their goals. The theory and models for confidentiality do not apply well to integrity and there is thus far no completely satisfactory proposed integrity model.

Major areas of agreement were expressed as the need for 1) research in the area of separation of duties, 2) research in the area of interfaces that preserve integrity, 3) integrity guidelines, and 4) ways to implement the Clark/Wilson access triples. (The access triples refer to the need to tie together user, program and data so that integrity is maintained during system operation.)

Major areas of conflict were 1) a definite opposition to creating at this time a data integrity evaluation criteria document parallel to the DoD Orange Book for confidentiality [TCSEC] since it would be premature and 2) a lot more discussion of Integrity Framework Elements and integrity definition(s). Section 5 summarizes the workshop Day-2 reports in more detail.

1. INTRODUCTION

1.1 BACKGROUND

In April of 1987, a paper entitled "A Comparison of Commercial and Military Computer Security Policies," written by David Clark of M.I.T. and David R. Wilson of Ernst & Whinney [CLARK87], was presented at the annual meeting of the IEEE Technical Committee on Security and Privacy. The paper suggested that previous work on data security had favored confidentiality policy in response to the needs of the defense establishment (see the DoD Trusted System Criteria document, also known as the Orange Book [TCSEC]) and that this was done at the expense of integrity policy required by the commercial and non-defense government sectors. The paper proposed an integrity policy based upon requirements for separation of duties, well-formed transactions, and audit trails.

The intense interest of the computer security community, created by this paper, was channelled into the invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), which was held at Bentley College in Waltham, Massachusetts on Oct. 27-29, 1987 and was sponsored by the IEEE Computer Society Technical Committee on Security and Privacy, the Special Interest Group on Security, Audit, and Control of the Association for Computing Machinery (SIGSAC/ACM), the National Computer Security Center at the National Security Agency (NCSC/NSA), and the Institute for Computer Sciences and Technology at the National Bureau of Standards (ICST/NBS) [now called the National Computer Systems Laboratory at the National Institute of Standards and Technology (NCSL/NIST)]. After a number of presentations, the workshop dissolved into five working groups, each discussing a different area of concern with respect to the Clark/Wilson model. The areas of concern consisted of Assurance, Granularity and Functions, Identity Verification, Auditability, and Correspondence of a System to Reality.

The workshop attendees agreed that this was a good beginning to build upon but that the Clark/Wilson model was not a sufficient foundation for an integrity policy. In fact, Clark/Wilson, in a post workshop paper, suggested that 'integrity framework' rather than 'integrity model' might be a better description for their work. One major concern expressed at the workshop was that computer security involves the triad of confidentiality, integrity, and availability and this piecemeal approach (i.e., addressing confidentiality, and now integrity) might not enable one to eventually encompass all three aspects of security simultaneously. However, the workshop did provide a valuable dialogue between 'scientists' and 'practitioners', and combined ideas about information systems from the business sector with concepts from the military and general security research sectors. NIST has published the proceedings of this workshop in its Special Publication 500-160 [NIST160].

At the close of the WIPCIS workshop, NBS [now NIST] agreed to nurture the integrity policy activity, and recommended a plan of action, including the organization and hosting of one or more follow-on integrity workshops. A data integrity working group (WG), chaired by Robert H. Courtney, Jr., was formed within the NIST Computer and Telecommunications Security (CTS) Council and many discussions on data integrity and data quality ensued within this group. These discussions were instrumental in formulating the subject matter for the first

follow-on integrity workshop organized and hosted by NIST. The WG members are listed in Endnote 3 in the Courtney Strawman Paper presented at the follow-on Data Integrity Workshop.

The follow-on workshop, an invitational Data Integrity Workshop, was held at NIST on Jan. 25-27, 1989, was co-chaired by Robert H. Courtney, Jr. and Zella G. Ruthberg, and is the subject of this document.

1.2 OBJECTIVES OF THE WORKSHOP

The Planning Committee for the follow-on workshop contained many members of the WG and was responsible for outlining what should be achieved at the workshop, what should appear in the Proceedings, and the Agenda for accomplishing these things. The following material was decided upon at a Dec. 8, 1988 meeting of the Planning Committee. The members of the Planning Committee were:

> Thomas A. Berson (Anagram), Carl Landwehr (NRL), Zella G. Ruthberg (NIST), David Clark (MIT), Steven B. Lipner (DEC), Dennis Steinauer (NIST), Robert Courtney (RCI, Inc.), William H. Murray (Ernst & Whinney), Anne Todd (NIST), Stuart Katzke (NIST), Donn Parker (SRI), Willis Ware (Rand), Stan Kurzban (IBM), Sylvan Pinsky (NCSC), Larry Wills (IBM)

It was agreed that the workshop should work on the ideas illustrated in Figure 1-1:

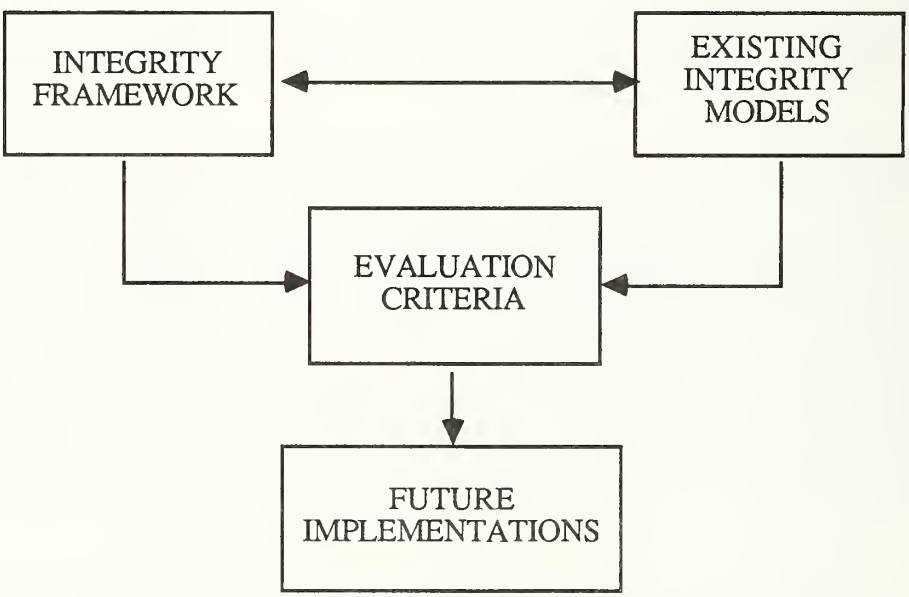


Figure 1-1 Workshop Subject Matter Relationships

Outlining the material to be addressed by the workshop in the form of a subsequent Proceedings, the Planning Committee arrived at the following:

I. Integrity Framework Elements

1. Introduction
   - Scope and Objectives
2. Concepts
   - Integrity, Quality, Value
   - Policy and Principles
   - Attributes of Quality
   - Roles of People
3. Basic Integrity Requirements
4. Quality Metrics

II. Implementation Requirements and Approaches
(emphasizing functionality and properties)

1. Operating Systems and Systems
2. Telecommunications
3. System Services (e.g., DBMS, TP Monitors, etc.)
4. Applications
   a) Highly Structured
   b) Not Highly Structured

Appendix: Implementations/Models - should be discussed in relation to the Integrity Framework developed in I.

The Proceedings would, at a minimum, then consist of the papers presented, the Integrity Framework developed in I, the Implementation Requirements and Approaches developed in II, and an Appendix on the Implementation/Models discussion relating these models to the Integrity Framework developed in I.

1.3 AGENDA OF THE WORKSHOP

The following Agenda for the workshop was then arrived at, based on the Proceedings outline in Section 1.2:

DATA INTEGRITY WORKSHOP
JANUARY 25-27, 1989
at NIST

AGENDA

Wednesday, Jan. 25, 1989

8:00am - 9:00am	REGISTRATION

PART I - INTEGRITY FRAMEWORK ELEMENTS

9:00am - 9:15am	Introduction - Workshop Objectives Dennis D. Steinauer NIST
9:15am - 9:45am	Some Informal Comments About Integrity and the Integrity Workshop (Strawman) Robert H. Courtney, Jr. RCI
9:45am - 10:15am	Evolution of a Model for Computer Integrity David R. Wilson Ernst & Whinney
10:15am - 10:30am	On Data Quality Viiveke Fak Linkoping Univ., Sweden
10:30am - 10:45am	Orientation for Part I Discussions Zella G. Ruthberg NIST
	Divide into Parallel Working Groups. Get Room Assignments.
10:45am - 11:00am	COFFEE BREAK
11:00am - 12:30pm	Part I- Parallel Discussions on Integrity Framework Elements
12:30pm - 1:30pm	LUNCH
1:30pm - 4:00pm	Part I- Parallel Discussions (continued)
3:00pm - 4:00pm	COFFEE BREAK (available at will)
4:00pm - 6:00pm	Joint Session - Reports from Groups
6:30pm - 8:30pm	CASH BAR & DINNER
8:30pm -	Group Chairs and Recorders Have Discussion to Arrive at Consensus on Integrity Framework Elements

DATA INTEGRITY WORKSHOP
JANUARY 25-27, 1989
at NIST

AGENDA

Thursday, January 26, 1989

8:30am - 10:00am — Consensus Report on Part I- Integrity Framework Elements
by One or More Group Chairs

PART II - IMPLEMENTATION REQUIREMENTS & APPROACHES
APPENDIX - IMPLEMENTATIONS/MODELS & THE INTEGRITY FRAMEWORK CONSENSUS

10:00am - 10:30am — Terminology, Criteria and System Architectures for Data Integrity
Ravi Sandhu
Ohio State University

10:30am - 10:45am — COFFEE BREAK

10:45am - 11:15am — Integrity Controls for Military and Commercial Applications
Robert R. Jueneman
Computer Sciences Corp.

11:45am - 12:15pm — Security Classes and Access Rights in a Distributed System
Roy. W. Jones
ICL Defence Systems, UK

12:15pm - 12:30pm — Orientation for Part II and Appendix Discussions
Zella G. Ruthberg
NIST

Divide into Five Discussion Groups on Five Different Topics.
Get Room Assignments.

12:30pm - 1:30pm — LUNCH

1:30pm - 6:00pm — Five Different Discussion Groups

1) Operating Systems and Systems
Steven Lipner, Chair

2) Telecommunications
David Clark, Chair

3) System Services, eg DBMS, TP Monitors
Grant Wagner, Chair
DBMS Integrity & Secrecy Controls
Rae K. Burns
Kanne Associates

DATA INTEGRITY WORKSHOP
JANUARY 25-27, 1989
at NIST

AGENDA

Thursday, January 26, 1989 (cont'd)

1:30pm - 6:00pm Five Different Discussion Groups (cont'd)

4) Applications - Highly Structured
Unstructured
William H. Murray, Chair
Transformation Procedure (TP) Certification
Maria Pozzo
UCLA
Steve Crocker
Trusted Information Systems

5) Implementations/Models - and the Integrity Framework Consensus
Carl Landwehr, Chair
Toward a Model for Commercial Access Control
Stan Kurzban
IBM Corporation

3:00pm - 4:00pm COFFEE BREAK (at will)

6:30pm - 8:00pm CASH BAR COCKTAIL PARTY

Friday, Jan. 27, 1989

8:30am - 10:00am Five Discussion Groups Continue
10:00am - 10:30am Reports of Conclusions Begin by Five Discussion Groups

---

10:30am - 10:45am COFFEE BREAK

10:45am - 12:30pm Reports of Conclusions are Completed by Five Discussion Groups

WORKSHOP ADJOURNS

Afternoon - Session Chairs and Recorders meet to start writing reports for Proceedings.

1.4 SUMMARIES OF PRESENTED PAPERS

Section 1.4 provides a concise report on the papers presented at the workshop. The Call For Papers (see Appendix C for text), which came out in mid-September 1988, briefly described the background and objectives of the workshop and referenced the Courtney Strawman Paper (see Section 1.4.2 for brief description and Appendix A for text), which defined data integrity and data quality. The Courtney paper was sent out in late September and stimulated responses in the security community in the form of submitted papers and requests to attend. All submitted papers were either selected for presentation (see Appendix A for texts) or placed in a pool of Position Papers (see Appendix B for texts). In addition, Position Papers were requested from all attendees prior to the workshop. The criterion for selection of presented papers was how well the paper fit into the objectives of the workshop (see Section 1.2 for objectives).

1.4.1 Some Informal Comments about Integrity and the Integrity Workshop, by Robert H. Courtney, Jr. (Strawman Paper)

The Courtney strawman paper had two major objectives: 1) to present a definition of 'data integrity' arrived at by his Integrity Working Group within the NIST Computer and Telecommunications Security Council and 2) to limit the scope of the workshop by stating ten issues for discussion. The presented definition of integrity was:

Integrity -- The property that data, an information process, computer equipment, and/or software, people, etc., or any collection of these entities, meet an *a priori* expectation of quality that is satisfactory and adequate in some specific circumstance. The attributes of quality can be general in nature and implied by the context of a discussion; or specific and in terms of some intended usage or application.

The ten issues in the strawman addressed the following concepts: 1) information and data are different; 2) data value and data cost are different; 3) the attributes of accuracy, completeness, and timeliness contribute to data value; 4) data value does not necessarily increase with data quality; 5) a cost-benefit relationship between data quality and data value should be established; 6) policy should guide attaining data integrity and data quality; 7) policy should reflect principles guiding conduct of functions under that policy; 8) there exists a large body of such principles from which to select for a particular application; 9) need to identify these principles for inclusion in a shopping list; and 10) need to understand appropriate roles relative to data quality and data integrity, i.e., Quality Control, Quality Assurance, Data Base Administration, the MIS Management, the functional area managers supported by the data, the internal audit functions, the external auditors, and the computer security group. For a discussion of the Courtney paper's definition of data integrity as well as others presented at the workshop, see Section 2.

1.4.2 Evolution of a Model for Computer Integrity, by David D. Clark and David R. Wilson

The original Clark/Wilson paper [CLARK87] defined integrity as "those qualities which give data and systems both internal consistency and a good correspondence to real world expectations for the systems and data." Because of this, integrity controls can never be strictly

internal to the computer, but must also include cross-checks with external reality. The computer system can be expected only to preserve the integrity once data has been externally verified.

Methods for assuring internal consistency should be based on 1) prevention of change, 2) attribution of change by binding data to its author, 3) constraint of change via certified Transformation Procedures (TPs), and/or 4) partition of change to insure separation of duties. Three key ways to relate a system and data to the world they represent are: 1) via an Integrity Verification Procedure (IVP) that not only ensures that a system is initially in a valid state, but also verifies the consistency between the data and external reality; 2) via a Transformation Procedure certification that assures a) the TP does what it is supposed to do and b) the TP specification really corresponds to the 'real-world' process it models; and 3) via Separation of Duty rules that assure a sufficient breakdown of tasks so that people with different motives and perspectives necessarily handle these component tasks.

Many of these principles can be carried forward into an integrity features list that could be incorporated in evaluation criteria for future computer systems. These are, in brief:

1) Data author label, Data change logs, Data integrity label
2) Support of access triple (user-ID, TP, (CDI_1,CDI_2,...))
3) Enhanced user authentication (for partitioned changes, without forgery, and without shared identity)
4) Control of privileged users - to enforce separation of duties
5) Application program control - to prevent administrative or technical corruption of a program
6) Dynamic separation of duty related TPs - some at the application level and some at the operating system level.

The paper then attempts to build on the existing Orange Book [TCSEC] requirements for disclosure levels C, B, A by mapping integrity features into this structure and then categorizing these features as Mandatory or Discretionary and Required or Optional. Much research needs to be done to achieve a satisfactory mapping. Finally, the paper states that the third piece of the security triad, i.e., denial of service, needs to be addressed for a complete security model.

1.4.3 On Data Quality, by Viiveke Fak

The author, in a very reasoned manner, discusses the words 'data integrity', 'security', 'confidentiality', and 'data quality' and decides to accept 1) the common usage for security that includes confidentiality, and integrity as independent components, and 2) the strawman paper definition of data integrity that relies on measuring data quality. She concludes that data quality has the measurable attributes of accuracy, completeness, timeliness, interoperability, and relevance, and discusses aspects of these attributes.

1.4.4 Terminology, Criteria, and System Architectures for Data Integrity, by Ravi Sandhu

The author accepts the strawman definition for integrity, but introduces the terms 'trust' and 'mandatory controls' in a manner different from the strawman paper and the Orange Book

[TCSEC]. He proposes using 'trust' as a binary term, applicable to active entities and synonymous with 'integrity'. He argues that 'trust' is not synonymous with 'confidence'. This view of 'trust' emphasizes two distinct issues in evaluating trust, functionality and degree of confidence. He proposes that 'mandatory controls' be used to mean controls based on properties of the object and/or the subject. Label-based controls are then a special case of this more general notion. Discretionary controls can be excluded by excluding properties based on identity of the subject and/or the groups to which it belongs. Otherwise, discretionary controls can be viewed as a very special case of mandatory controls.

With this perspective, criteria for evaluating trusted systems must clearly separate the issues of functionality and degree of confidence. While some joint progression may be inevitable, it is inappropriate to couple these two issues too tightly. Also, the criteria need to be finer than those found in the DoD Criteria document [TCSEC].

With regard to trusted systems architecture, experience has shown that trusted code is needed above the security kernel so that mandatory controls can be by-passed in certain legitimate circumstances. Policy and mechanism should be separated in the kernel. Policy should be specified by static policy tables which can only be changed, under careful control, when the system is built. A built-in notion of types should be provided for specifying these policies.

Much research in the above areas is needed. It appears that mandatory controls for integrity will be an order of magnitude more complex than label-based mandatory controls for non-disclosure except for the covert channel problem which makes non-disclosure such a difficult problem.

1.4.5 Integrity Controls for Military and Commercial Applications, II, by Robert R. Jueneman

This is a very extensive paper in which a system of integrity controls is presented that is intended to be suitable for both commercial and military applications. The implicit context of the Clark/Wilson paper [CLARK87] was that of a monolithic mainframe computer processing the traditional MIS programs (e.g., accounts receivable and payable, inventory control, payroll) and did not cover such critical activities as computer program development, CAD/CAM, robotics, nuclear reactor control, hospital patient monitoring, air traffic control, embedded tactical applications, the Strategic Defense Initiative, or even simple word processing. In addition, there was no discussion by Clark/Wilson of the requirement for Electronic Data Interchange of purchase requisitions, invoices, or contracts between disparate organizations which would necessitate the use of digital signatures that would stand up in court if challenged. This paper is somewhat biased toward single user dedicated workstations embedded in very large and dynamic networks of interconnected Trusted Computing Bases (TCBs) which communicate over a virtual network of arbitrary non-secure media. The paper assumes the TCBs provide the equivalent of a B2 level of trust in accordance with the DoD Criteria document [TCSEC] and have an embedded cryptographic functionality which can be used to protect both the secrecy and integrity of data stored locally or exchanged via a network.

Integrity threats that are present in a large and dynamic network of TCBs are examined and the point made that the DoD Criteria [TCSEC] address only the TCB and not programs and data. After much discussion, the conclusion is drawn that a containment mechanism rather than extensive Access Lists is the more feasible approach to prove the negative proposition that no other program or agent, either internal or external, could possibly alter a Constrained Data Item (CDI), except as permitted by the list of the Clark/Wilson access permission triples. The class of containment mechanisms suggested is that of detection mechanisms which would invalidate the data if such a change were to occur, plus the use of normal recovery/restoral mechanisms to repair the damage.

The definition of integrity proposed is adapted from the Courtney/Ware definition found in the strawman (see Appendix A) and is:

> "Integrity is a property of those processes (subjects) and information (objects) that meet an a priori expectation (specification) of a level of quality that is considered acceptable for a particular application."

A discussion of this definition leads to a description of a metric for integrity, i.e., it must consist of 1) a specified set of rules and 2) a measure of trust or confidence, 3) that is determined by an acceptable authority.

The integrity requirement generates a need for having valid data operated on by a valid transformation process (TP) that results in valid data. This leads to the introduction of the concept of an Integrity Domain - the set of allowable inputs to a process together with the syntactic and semantic rules used to validate or reject these inputs and optionally produce one or more outputs. The conclusion is then drawn that the integrity of a process is probabilistic, not binary, and can only be defined relative to a specified integrity domain.

An extensive discussion of integrity threats and countermeasures leads to an enunciation of thirteen rules, fashioned after but more comprehensive than those found in the Clark/Wilson paper. These are claimed to be necessary and sufficient for integrity concerns in a networking environment. The rules briefly are:

Rule 1 Immutability - Must have the ability to verify that contents of an object are unchanged.

Rule 2 Attribution - Objects must be attributable to an authorized user or "originator."

Rule 3 Validation - Trusted Transformation Procedures (TPs) shall either validate or reject objects in their integrity domain.

Rule 4 Certification - TPs shall be certified by an approval authority to perform their function correctly.

Rule 5 Separation of Duty - Must be able to constrain who can execute a trusted process and/or create or modify certain data so that change is partitioned among users.

Rule 6 Identification - All users of a system shall be uniquely enrolled into and identified to the system by a trusted registrar.

Rule 7 Authorization - An individual's clearances, rights, privileges, and authority shall be authenticated.

Rule 8 Trusted Date/Times - TCB shall provide a trusted Date/Time function that is maintained within N seconds of Greenwich Mean Time (GMT).

Rule 9 Trusted Sequencing - TCB shall provide sequencing mechanisms sufficient to assure that programs are applied to data in prescribed order.

Rule 10 Auditing - TCB shall at a minimum create an audit log entry for each file imported into the system. This entry should contain comprehensive data on the file, i.e., user, process, date, time, TCB used, cryptographic checksum, etc.

Rule 11 Pedigree - TCB shall provide the option to capture in an integrity label the audit log level (pedigree) of the file.

Rule 12 Provenance - TCB shall provide a means for a user to indicate his/her approval of the contents of an object and enter associated data (e.g., description of object's contents, associated documentation). This information is called the provenance and shall be recorded in both the audit log and the integrity label.

Rule 13 Human-Readable Output - TPs shall be certified to correctly display and print the desired human-readable output. If other non-certified programs intervene between the TP and final display or printout, the human user must be responsible for the integrity of the output.

A set of implementation mechanisms for these rules are then proposed and discussed at length. These include such considerations as integrity labels, a hierarchical integrity level (defined as an estimated probability that the object met defined syntactic and semantic rules), a set of non-hierarchical integrity categories or attributes corresponding to a set of integrity domains, a reference monitor for integrity, the dynamic mapping of integrity categories, a set of global integrity attributes (e.g., VERIFY, RECORD, PEDIGREE, DELETION-SENSITIVE, and the digital signature related attributes of NON-REPUDIATION and NOTARIZED), and integrity covert channels.

It is concluded that a system of Mandatory Integrity Controls consisting of the Biba hierarchical integrity policy with integrity categories and multilevel integrity-trusted subjects, plus Discretionary Integrity Controls which allow a subject to specify which objects are to be accepted or believed, implemented on a TCB that meets Orange Book requirements for B2 with respect to security, can provide integrity controls well suited to the networking environment. The author believes that this paper represents a reasonable consensus approach of work already done in the integrity area and that it is "do-able" using current technology.

1.4.6 Security Classes and Access Rights in a Distributed System, by Roy W. Jones (presented by Tom Parker)

The paper describes a general design for a secure system using a distributed network of computing resources and does so with a combination of set theory and logic. It provides an appropriate notation which enables the requirements of the secure system to be related to its design in a mathematically more precise manner. In order to accomplish this, the paper defines a number of terms and ideas that can be used in secure system design (e.g., class, classification, access right). Finally, it describes the Clark/Wilson integrity model in these terms and notation.

Some of the key notions developed are:

1) Definition of a generic mathematical form for a security class
2) Definition of a total security class for a particular system
3) How to classify all the components (data items, computing resources, end users) of a system into subclasses
4) How to assign access rights to a class

An active component is defined as one which may operate upon other components. All others are classed as inactive. Each active component has a clearance which is a class with associated access rights. A security policy can then be described as a set of rules which associate classification class with maximum permitted clearances.

The value of this work is that it presents a model for describing a secure system in a logical manner and uses a mathematical notation to do so. A more rigorous description of the model is needed as well as the development of real world examples in detail.

1.4.7 DBMS Integrity and Secrecy Controls, by Rae K. Burns

This paper briefly discusses three areas overlooked in current discussions on data integrity. The first is that the Clark/Wilson paper is addressing the same domain as database management systems. The Clark/Wilson principles of well-formed transactions and separation of duties, which lead to notions of constrained data items (CDIs), transformation procedures (TPs), and integrity verification procedures (IVPs) are not directly found in data communication systems, operating systems, or even applications. Database management systems, on the other hand, have vehicles for expressing all three of these concepts (CDIs, TPs, and IVPs).

The second area is the notion that application secrecy requirements and integrity requirements are interrelated and must be handled in a unified context. A fundamental relationship between classification constraints and integrity constraints is that a classification constraint is basically an integrity constraint that applies to a secrecy label attribute rather than a data value attribute. It is not feasible to isolate secrecy and/or integrity semantics from the overall application semantics.

The third area involves the inappropriate separation of discretionary access controls for operations that modify data (integrity) from those controls for operations that only read data (secrecy). As an example, within many database applications, *ad hoc* queries are no more appropriate for read-only transactions than they are for modification of the database.

1.4.8 Work-In-Progress: Transformation Procedure (TP) Certification, by Maria Pozzo and Steve Crocker

A primary component of the Clark/Wilson Model [CLARK87] is the Transformation Procedure (TP), which corresponds to the notion of a well-formed transaction. A TP is a program that has been certified to change the data it manipulates in constrained ways. This paper discusses a proposal for a TP certification mechanism based on formal specification and verification techniques. The general idea is to build a filter program which examines a potential TP to determine 1) if it manipulates only those Constrained Data Items (CDIs) that it is

supposed to manipulate and 2) if it does so in a manner that is consistent with the system's integrity policy for CDIs.

The issues in building this filter fall into three classes: specification, analysis, and acceptance. Separation of the integrity component of what the program does from general program correctness issues simplifies the specification process under consideration. In the context of integrity, a primary concern is to restrict the data that a program can manipulate. This less general form of specification is called a restriction. The filter then analyzes a program to determine if the program stays within its restriction. The next step is to decide whether the restriction permits actions that are acceptable according to the system's integrity policy. A program then becomes a TP if it can be shown that it implements the integrity policy associated with the CDIs it manipulates. The paper then discusses some of the ways that the specification might be constructed (e.g., list, pattern match, complex rule, type).

In the analysis area, the paper recommends that a TP use software engineering techniques to limit its actions so that open-ended capabilities such as dynamically generated code that it branches to or the ability to make operating system incisions do not occur. In the acceptance area the conclusion is drawn that it is not possible to draw up a single integrity policy for a wide range of information and applications and that such policy is most likely application dependent.

Two examples are then presented: one in an accounting domain where CDIs are highly structured data items; and a second from the viewpoint of computer virus protection where the CDIs are generally unstructured entities. Current work underway is investigating the feasibility of applying the filter approach to virus protection.

1.4.9 Toward a Model for Commercial Access Controls, by Stan Kurzban

This paper presents a set of objects that are suitable for use in rigorous descriptions of access control software to meet commercial requirements. The most important of these requirements can be expressed in terms of the following four principles (abbreviated):

1. Least Privilege (LP) - People must be authorized to do all and only what they must do to perform their assigned tasks.
2. Separation of Duties (SD) - If a set of acts can jeopardize the organization, then multiple individuals with conflicting motives must be involved in performing sets of acts.
3. Ease of Use (ESU) - The easiest way to do something should always be the safest way as well.
4. Adequate Invulnerability (I) - Deliberate circumvention of security measures will be demonstrably less attractive than other means of achieving the same objectives.

Clark/Wilson [CLARK87] used these principles to derive their model for commercial computer security and their work and this paper suggest the following access control fundamentals (abbreviated):

1. The facilities of access control must be available to application software (LP).
2. Access control software must permit the aggregation of both users and resources into groups (ESU).
3. Users must be able to prove their identities easily (ESU) as well as with certainty (I).
4. The separation of administrative tasks by scope and responsibility (SD) is vital to placement of small burdens on administrators (ESU).

The users of systems can be classified as Application Developers, Application Users, and Administrators. The paper enumerates their corresponding security needs. The model's objects of interest for commercial access control are User Profiles, Group Profiles, Resource (and Resource Set) Profiles, Programs, Processes, and System-Wide Security Data. The paper describes each of these objects.

The next step would be to map these objects to some formalism that would permit the development of a model. Such a model would then enable one to develop an approach to rating access control products, establish an architecture for access control, verify the functioning of an access control product, and even automate such verification via theorem provers.

1.5 GUIDE TO REPORT'S CONTENTS

This document takes the reader through the entire process of organizing, holding and reporting the results of this Invitational Workshop on Data Integrity. Section 1 gives the reader background on the work on integrity policy that preceded this workshop, what the objectives of this workshop were, how the agenda was developed, and, via summaries, the major ideas presented by the Strawman and the eight papers presented at the workshop. The Call for Papers that lead to the selection of the presented papers can be found in Appendix C.

Since a consensus on the definition of data integrity was not reached, Section 2 was written to present some thoughts on the various views on the definition of this term. Section 3 contains summaries of the five Day-1 discussion groups' deliberations on Integrity Framework Elements and briefly presents the consensus arrived at for data quality related policy/objectives and common mechanisms. These consensus items were to be used as the basis for Day-2 deliberations. Section 4 contains the reports by the five groups on their Day-2 discussions.

The Day-2 reports each contain conclusions, recommendations, and statements of major issues that need further work. Section 5 attempts to summarize the results of the workshop efforts and points towards areas needing more research. Section 6 contains an integrated reference list derived from all the group reports.

Appendix A contains copies of all the papers that were presented at the workshop. As part of the admission to the workshop, invitees were asked to write a brief position paper, if possible. Appendix B contains all of the papers that were submitted for this purpose. Appendix D provides the reader a detailed attendee list.

2. THE DEFINITION OF INTEGRITY

2.1 INTRODUCTION

The definition of data integrity is not a new subject. FIPS PUB 39 [FIPS39], the **Glossary For Computer Security Systems Security**, defined data integrity as "The state that exists when computerized data is the same as in the source documents and has not been exposed to accidental or malicious alteration or destruction." This definition has grown narrow, as the capabilities of computer systems evolved. It is no longer acceptable to limit our discussion of data to source documents. The integrity of data which has been processed, modified or restructured is of importance in modern computing. So, this definition must be superseded by a more comprehensive definition.

[Note: This section refers to many documents contained in the Appendices. Those documents are noted by author or title; any other reference is in the References, Section 6]

The subject has, however, languished in relative obscurity as the field of Computer Security grappled with the more immediate problem of confidentiality. Confidentiality is the primary concern in the handling of classified information. A set of criteria was developed by the U.S. Department of Defense for the classification of computer mechanisms to provide control of classified information. This set of criteria is contained in a popularly DOD publication known as the Orange Book [TCSEC]. The field has now turned to the problem of control of unclassified information, where it is agreed that data integrity is the primary concern.

This grappling with the problem of integrity began with an attempt to use the types of access controls described in the Orange Book for enforcement of the Biba integrity closure rules. Most notably, Steve Lipner of DEC presented the paper "Non-Discretionary Controls for Commercial Applications" [LIPNER82] at the 1982 IEEE Symposium on Security and Privacy. Additional work was done in the area by other researchers, but computer security specialists have, for the most part, concluded that these controls are simply inappropriate for the task. (For a dissenting opinion, see William Shockley's position paper "Importance of Mandatory Integrity Controls" in Appendix B.)

While there is consensus that the FIPS 39 definition is inadequate, there is no consensus upon what would be adequate. Since the publication of FIPS 39 in 1976, "data integrity" has been used to mean many and varied things. In general, the phrase has been used to describe the security policies of commercial (unclassified) systems. The requirements of those policies and the controls used to implement them are subject to some dispute. As a result, there is no agreed definition or model for data integrity ready to replace it.

2.2 CLARK/WILSON DEFINITION OF INTEGRITY

The ground swell of interest in data integrity was spurred by the recent paper "A Comparison of Commercial and Military Computer Security Policies" [CLARK87] presented at the 1987 IEEE Symposium on Security and Privacy by David Clark and David Wilson. While the Clark/Wilson paper did not state a specific definition for data integrity, it did

introduce a more formal model of data integrity. The Clark/Wilson paper concluded that a distinct set of security policies relating to data integrity were required in the commercial sector. In addition it concluded that these policies, and the mechanisms for their enforcement, were disjoint from the criteria presented in the Orange Book.

The Clark/Wilson model defines the data items to which the model must be applied as "Constrained Data Items" (CDIs), and defines two classes of procedures, "Integrity Verification Procedures" (IVPs) and "Transformation Procedures" (TPs). The model assumes that there are a set of valid states for the collection of CDIs in the system. If the CDIs are in a valid state, then the system meets the integrity requirements. In this model, the IVP is used to verify the state of the CDIs (eg, in an accounting system, an IVP would balance and reconcile the books). A TP is a procedure that will modify a set of CDIs, transforming them from one valid state to another. In this way, a system validated by an IVP and acted upon by TPs will remain in a valid state. This model describes the progression of data through a system and defines its integrity at every point. Implicitly, this model describes data integrity as binary, and applies this to a system as a whole rather than specific data items.

A second paper by Clark and Wilson, "Evolution of a Model for Computer Integrity" [CLARK89] presented a definition for computer integrity. This definition stated that computer integrity is related to both an internal and external consistency standard. External consistency means that the data conforms to the real-world situation it describes; internal consistency states that any modifications to the data have been performed in a controlled manner.

2.3 STRAWMAN DEFINITION OF INTEGRITY

The Strawman paper detailed a definition of integrity that was in some ways a parallel of that implied in the Clark/Wilson model. This definition incorporates three basic concepts: data quality; expectations of quality; and data integrity. Data quality is a new term introduced to describe such attributes as accuracy, timeliness and completeness. Data quality is an analog (continuously valued) function. Expectations of quality are the minimally acceptable value of data quality (as defined by the user). Finally, data integrity is defined as a purely binary value determined by the data quality's relationship to expectations of quality. The data has a certain degree of quality, and the user has some expectations of quality. If the data quality equals or exceeds the expectations of quality, the data has integrity; otherwise it does not.

Certain features of the strawman definition were especially controversial; most notably, the introduction of a new term "data quality" and the proposal that data integrity is a purely binary idea.

2.4 INTEGRITY DEFINITIONS: THE WORKSHOP PAPERS

The responses fell into several overlapping groups. One group of papers assumed Clark/Wilson or the strawman and proposed methodologies for maintaining data integrity in a system based on that definition. (These papers will not be discussed further in this section.) Another group commented on the model proposed by Clark/Wilson or the strawman definition, and proposed alternatives or modifications. This group may be further divided into those addressing the binary nature of integrity in the strawman and those viewing data integrity as

a data correctness issue. A third group outlined issues that they felt should be addressed at the workshop, and several additional definitions of integrity were also proposed. A fourth group chafed noticeably at the constraint that they address data integrity, feeling compelled to broaden or narrow the focus. A few authors consciously avoided the strawman or Clark/Wilson paper to constrain themselves to their instinctive opinions, rather than be influenced by those positions. Finally, while the Clark/Wilson paper's basis is the idea that current methods cannot be reasonably applied to the problem, one author (Shockley) argues that those methods can be reasonably applied now. He argued that their imperfection for the task should not rule out their use until more suitable methods are devised.

2.4.1 Modified Definitions

Robert Jueneman's presented paper "Integrity Controls For Military and Commercial Applications, II" includes a detailed definition for data integrity that is based on the same concepts as the strawman definition. This definition is based on a priori expectations of quality, as in the strawman. However, integrity is measured as two components. [The first component is scalar and the second is hierarchical. The scalar component specifies the integrity category corresponding to the integrity domain.]

Ravi Sandhu presented the paper "Terminology, Criteria and System Architectures for Data Integrity", in which he proposed an alternate definition for trust. This new definition did not alter the relationship of integrity and confidence, but rather altered the relationships between trust, integrity and confidence. Sandhu's definition of trust is binary, and is evaluated accordingly by comparing the expected and actual level of confidence. This new definition formed the basis for his argument that functionality should be separated from assurance in criteria for the evaluation of trusted systems.

2.4.2 Is Data Integrity the Appropriate Topic?

The workshop had been restricted to the subject of data integrity. In retrospect, this restriction seems to have posed a problem in and of itself. The strawman defines data integrity in terms of data quality. Data quality is undefined itself. On the other hand, data integrity is but one piece of the larger puzzle of commercial security requirements. Those requirements are the subject of continuing discussion as well. Perhaps the scope should have been wider, or more narrow, but not in the middle ground that was chosen.

Some authors felt that data integrity could not be discussed in an entirely isolated fashion. Others felt that it was too early to discuss solutions, and proposed questions instead. One author found it necessary to narrow the scope and address quality issues only.

Both Donn Parker and Nander Brown found it necessary to describe the larger world to obtain an appropriate framework to define data integrity. For Parker, the definition of integrity is based upon your role in the larger scheme of things. Integrity is binary and invariant as required by clients; integrity is variable from the view of the security specialist and those attacking the system. For Brown, integrity is the appropriate measures required to meet the objectives of the system. In "Achieving Integrity In Automated Systems", Brown defines data integrity, processing integrity, software integrity and systems integrity. The definition of data

integrity includes the notions of accuracy, completeness and timeliness. (These are all QUALITY attributes in the strawman.)

Stewart Kowalski states that integrity is a system concept; data can only be correct or incorrect. David Bonyun's paper presented the status of work on integrity in ISO standards. The ISO definition assumes multiple perspectives for integrity depending upon application (examples are mainframe operating systems vs. database vs. communications systems).

Thomas Chen did not explicitly define data integrity in "Other Informal Comments...", but listed several requirements for a system to have/maintain integrity [that a definition for data integrity should enforce]. Jan Kruys chose to pose questions, rather than propose answers, in his position paper.

Eva Sparr, on the other hand, narrowed the scope in her position paper. "Data Or Information Quality" presents a model for data quality where there are two major attributes: relevance; and validity. Relevance is a measure of the appropriateness of the data in relation to the user's problem. This is a function of such factors as "what the data involves, under what circumstances they were gathered and the number and type of errors the data contains." Relevance is a binary attribute; data has relevance to a particular application. Validity is similar to the Clark/Wilson notion of external correspondence. Validity is a continuously valued function. Data may be valid but not relevant, or may have low validity but still be relevant.

The Strawman definition of data integrity may easily be extended for the Sparr model of data quality. The expectation value for quality would simply be applied to the validity attribute. For data to have integrity, the data must have relevance and the validity attribute must be equivalent to, or in excess of, the specified expectation.

2.4.3 Is Integrity Binary?

Several position papers, including those by Theodore Lee, Lee Ohringer and Sig Porter took issue with the binary nature of the strawman definition. Theodore Lee of TIS questioned the idea that integrity could reasonably be described as binary. Lee Ohringer liked the concept of quality and integrity, but still felt integrity should not be binary. Sig Porter expressed the opinion that multi-leveled integrity is often a valid approach. Stelio Thompson-Sittas proposes that integrity be continuous, rather than a multi-valued discrete function.

2.4.4 Is It Possible To Converge On A Single Definition?

Several authors expressed doubt that a single acceptable definition could be found. Thomas Chen states 'Since integrity comes in "different flavors", a proliferation of definitions is somewhat unavoidable' Sig Porter feels that we should avoid converging on a single definition at this time. The many definitions have evolved over time because each definition has a basis in certain types of systems. Rather, it is important to understand the differences and similarities between the definitions. Once these are well understood, a single definition may fall out.

2.4.5 Perhaps The Strawman Is Unnecessarily Complicated

There were also a few authors who seemed to feel that the strawman definition for data integrity involves unnecessary complication. Marv Schaefer discussed the importance of the Biba rules and "the fundamental importance of considering integrity as a data correctness issue". For Stewart Kowalski, data is correct or incorrect, period. Integrity is a systems issue. It is probably worth noting that this "back to basics" approach is a return to the very general FIPS 39 definition (albeit without the restriction to data entry).

2.4.6 Is Integrity Stateless?

State, or lack of state, is an important distinction in the classification of computing models. There are important ramifications for the implementation of the model. A stateless model requires no memory of previous actions or occurrences. The Clark/Wilson model and Strawman definition are both stateless models. However, it would be possible to build both IVPs and expectation mechanisms which used history to maintain, adjust and apply themselves.

Kurt Meiser expressed the idea that maintenance of data integrity was a "status quo" problem. Data has integrity if "the data presented by the system ... is good (identical with the data received by the system during previous operations)". This is an important departure from other definitions presented - in all other definitions, integrity is stateless, but here integrity is stateful; i.e, history of presented data is required to calculate the integrity value.

Stelio Thompson-Sittas proposes an alternate definition in the position paper "Definitions and Concepts Of Data Integrity". Thompson-Sittas defines integrity as "The ability of an element, or system of elements, (whether data, programs, hardware, personnel, etc.) to retain its attributes, and associations and guard against non-intended changes as the element proceeds through its lifecycle." As in Meiser's definition, integrity is stateful, so history is required. It differs in that it requires a history of the data itself.

It seems clear that situations might require either type of stateful Integrity. There is obvious power in definitions that are explicitly stateless, but are flexible enough to do so implicitly, such as Clark/Wilson or the strawman. Either stateful model (Meiser or Thompson-Sittas) could be supported by a stateless model if the expectations could be dynamically adjusted. This would allow the mechanisms for checking integrity to be very simple, but the mechanisms for selecting the expectation values might be quite complex. The ramifications on systems design (requiring maintenance of expectation mechanisms as application or data dependent) are quite extreme, though. It is unclear where this functionality would be appropriate.

2.4.7 A Mathematical Model

The search for an appropriate definition continued into the workshop itself. The first day's working groups re-visited the subject. Thomas Berson's group approached the question from a new tack, and created a mathematical definition for "acceptability of data". Acceptability of a data object was a function of three ordered lists of length n. The ordered lists represented the a priori expectation of quality, measured quality and confidence,

respectively, for each of the n attributes. The acceptability is computed by summing the confidence factor for each attribute whose measured quality met or exceeded the a priori expectations.

2.4.8 Consensus Items

After the first day of the workshops, the results were examined for areas of agreement. The attendees were able to agree on a set of policy objectives and a list of common mechanisms used for policy enforcement (see Section 3.2). There was no agreement about the definition itself, but the objectives and mechanisms imply support for two definitions of data integrity: accuracy and the Clark/Wilson model. The policy objectives included notions of privilege and accuracy of data as well as the Clark/Wilson concepts of constrained data items and well-formed transactions. The mechanisms noted were those for access control and the Clark/Wilson enforcement mechanisms (Constrained Data Items and trusted Transformation Processes). (Audit trails and auditability were also mentioned; this does not affect the definition chosen for integrity.)

2.5 THE POST-MORTEM/CONCLUSIONS

The most important conclusion to be drawn from this compilation of papers and working group notes: don't draw too many conclusions about the appropriate definition for data integrity just yet. All of the definitions incorporate the concept of accuracy; but all of the definitions include additional concepts as well. It isn't even clear that a single definition will suffice. A single definition may be so broad that it is not implementable. Multiple specialized definitions would be easier to implement but might not be interoperable.

On the other hand, don't try to propose an Integrity interpretation of the Orange Book as a stop gap measure - the mechanisms are just not applicable. It may be necessary for lab implementations to be constructed with various definitions, to see how they hold up under real-use conditions. In the mean time, papers addressing integrity issues should present or reference a definition of integrity applicable to that paper.

3. INTEGRITY FRAMEWORK ELEMENTS (DAY 1)

3.1 SUMMARIES OF FIVE GROUPS' DISCUSSIONS

3.1.1 Operating Systems Working Group (Group 1)
Prepared by Tom Berson

Tom Berson (Anagram) Chair
Steven Lipner (DEC) Recorder
Tom Chen (Wang) Ken Eggers (MITRE/McLean) Stu Katzke (NIST) Jan Kruys (NCR) Kurt Meiser (C&L) Tom Parker (ICL) Paul Peters (NCSC) Sig Porter (Consultant) Damian Saccocio (NRC) Gary Smith (GMU) Steve Welke (IDA)

1. Framework Scope and Objectives

The group decided that a suitable objective for the framework being established by this workshop was for it to serve as a target to encourage further research and refinement.

It was strongly agreed that it would be inappropriate to give the current framework any sort of status beyond that of target. Members expressed the view that the evolving framework was young and untried. They were unwilling to say that it was a correct framework, much less that it was the correct framework.

The group viewed with alarm its perception that NIST was rushing to institutionalize the results of this workshop through some sort of legislation (e.g. guidelines or criteria). The manufacturing members were particularly concerned about the potential for unforeseen and unplanned interference between a putative Integrity Criteria and other government criteria, such as the Orange Book [TCSEC].

2. System Security

The group observed that the notion that

> information system security is comprised of three components: confidentiality; integrity; and availability

has been around a long time, is widely-held and well-accepted, is taught by many FIPS PUBs and texts, and is a useful concept.

We concluded that it would be a disservice to weaken or destroy this concept by introduction of a definition of integrity which does not take its place within this triad. In particular, we felt that the Courtney/Ware definition, based upon *a priori* expectation, was overly broad for something called "integrity" as it might be usefully applied also to confidentiality and availability.

We did feel that while Courtney/Ware is not what we call integrity, it does capture the useful concept of acceptability.

3. What is Integrity?

We decided that our understanding of integrity includes

(a) constraint (including prevention) of change or creation;
(b) attribution to person or process of change or creation;
(c) authentication of origin;
(d) internal validity;
(e) guarantees of correct sequence

We observed that this notion is similar to that described in the generalized Clark/Wilson model.

4. Data Quality

Courtney and Ware instruct us to consider the acceptability of data for a particular application by taking measurements of their quality and comparing these with our expectations.

Quality, Q, is comprised of measurable quality attributes q_1, ..., q_n. Note it is important that the q_i be measurable, for if they are not then no calculation can be based upon them.

Examples of data quality attributes include: accuracy, pedigree, self-consistency, validity, and correspondence with other components of the system or with the "real world."

5. Acceptability of Data

We generalized the notion of acceptability of data as follows:

For a set of n quality attributes

Let $QP = qp_1, ..., qp_n$ be the a priori expectation of quality,

$Q = q_1, ..., q_n$ be the measured quality, and

$C = c_1, ..., c_n$ be the confidence in each measure.

Then, in general

$$\text{Acceptability} = F(QP, Q, C). \qquad [1]$$

Note especially that our generalized acceptability is potentially a continuous function, may give different weights to different quality attributes, and may take note of the confidence on a measurement.

Courtney/Ware acceptability is a special case of equation [1] in which the function is binary (i.e. either TRUE or FALSE), all quality attributes receive equal weight (each has veto power), and confidence in measurement is not considered.

$$\text{Courtney/Ware} = \underset{i=1}{\overset{n}{\text{AND}}} (q_i >= qp_i) \qquad [2]$$

While equation [2] may be an appropriate data acceptability measure for some applications, we saw no reason to believe that it is universally appropriate. We encourage those concerned with measuring acceptability of data to use the generalized form of equation [1] and to formulate an F which is appropriate to their needs.

3.1.2 Telecommunications Working Group (Group 2)

Prepared by Z. G. Ruthberg

David Clark (Chair)
Ted Lee (TIS), Eugen Bacic (Comm. Sec. Est.), Viiveke Fak (Linkoping U.), Howard Johnson (Info. Intell. Sci. Inc.), Stewart Lee (U. of Toronto), John Thurlow (Exxon), David Wilson (Ernst & Whinney)

1. Why is meeting important?

Basically, we do not trust our systems. Yet, such areas of endeavor as computerized medical systems and white collar productivity require systems with integrity. Loss of integrity in medical systems can lead to personal injury and even loss of life, while loss of integrity in white collar activities such as business records can impose serious and sometimes irreversible damage on organizations that are dependent on their computerized records.

2. A Policy for Integrity

A system maintains integrity when change occurs, if and only if that change is appropriate. One form of appropriateness is that which tracks real world expectations for completeness, accuracy, timeliness, consistency, interoperability, and/or relevance of the system and its data. The current state of the system and data should also be consistent with an earlier state. An organization's integrity policy should embody these concepts.

3. Notions of Universal Mechanisms for Integrity

The following types of mechanisms may be necessary and are certainly useful for achieving integrity:

a. Application Specific Reference Monitor
b. Trusted Computing Base (TCB)
c. Detection and Recovery vs. Prevention

d. Individual Accountability
e. Separation of Duties
Dual Control
Least Privilege
} Redundacy

4. Assurance

As with the assurance of higher levels of confidentiality of data, higher levels of data integrity require further controls. Work needs to be done to enumerate these.

3.1.3 System Services Working Group (Group 3)

Prepared by Sylvan Pinsky

Sylvan Pinsky (NCSC) Chair
Grant M. Wagner (NCSC) Recorder
Deborah J. Bodeau (Mitre), David A. Bonyun (AIT), Rae K. Burns (Kanne Assoc.), Terry Mayfield (IDA), Roger L. Miller (IBM), Lee Ohringer (Dept of Treas.), Carl Pabst (Touche Ross Int'l), David Rosenthal (Odyssey Res. Assoc.), Marvin Schaefer (TIS), William R. Shockley (Gemini Comp. Inc.), Gary Smith (George Mason U.)

We immediately decided to focus on the technical meaning of integrity instead of the "English" definition. We felt that there was a need to include the notion of data properties; however, it was necessary to do this with respect to some context. In particular, integrity must be viewed in terms of a specific application and then the meaning of data properties will be examined within the context of the application. We also attempted to separate integrity from confidentially. This was not entirely possible; sometimes there was a dependency between these concepts.

Our limited discussion of Bob Courtney's definition resulted in agreement that his definition was a good basis to initiate deliberations on database applications. The only area of contention concerned Bob's assertion that integrity was "binary". Some members of our group felt it was binary; however, several had difficulty accepting the suggested binary nature of integrity. I believe that our group's acceptance of Bob's definition and the lack of agreement concerning the binary nature was typical of the entire group's feeling.

Early discussion was centered on the perceived threats to systems or applications involving integrity. We came up with the following list:

THREATS
Tampering
Fraud
Misappropriation
Misuse (Inappropriate use with respect to application)
Loss
Damage
Delay

Errors
Contamination

We were not completely satisfied with this approach, so we went to an Input/Output process approach:

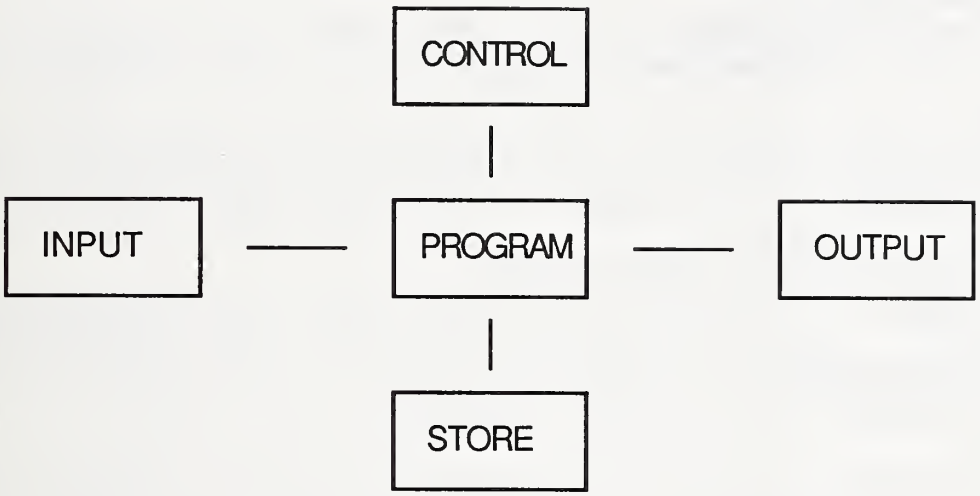


Figure 3-1: Input/Output Process Approach

To understand this approach better, we began to examine integrity requirements, quality attributes, etc. and characterized the following concepts:

QUALITY ATTRIBUTES
Accuracy (correctness or precision)
Pedigree
Self Consistency
Validity
Correspondence with other components of system
Timeliness
Completeness
Consistency among elements
Relevance
Interpretation
Robustness (relating to redundancy)

INTEGRITY REQUIREMENTS
- Internal Consistency
- Disclosure vs. Undisclosure
  - MLS DBMS (secret vs. unclassified views)
- Accuracy
- Completeness
- Well-formed Transactions
- Permissions and Authentication
- Monitoring (eg. distributed systems)

CONTROLS
- Separation of Roles
  - supervisory
    - content or value
  - authorized interruption (DBMS rollback)
    - overrides

CONTROL STRUCTURES
- Separation of Roles
- Monitoring
- Automated Interrupt
- Permission and Authorization
- Constraint to be enforced
  - View Mechanisms
  - Referential
- Configuration Management/Controls and Updates

Also, with respect to the potential dependence between integrity and confidentially and availability of services, we felt that a "layered" approach may be helpful to discuss the interaction between application specific integrity concerns and the mechanisms provided by the operating system environment. We felt that a "layered set of abstract machines" could be useful in discussing successful refinements of the integrity concepts. We were looking at this in a fashion similar to the "assured pipeline" approach used by Earl Boebert for LOCK.

We also categorized integrity failures in the following manner:

Types of Integrity Failures	Degree
Missing	Known
Wrong	Unknown
Extra	

3.1.4 Applications Working Group (Group 4)

Prepared by Karl Krueger

Dennis Steinauer (NIST) Chair
Karl Krueger (World Bank) Recorder
Nander Brown (SBA), Peter Capek (IBM), Dale Miller (Irongate), Bill Murray (Ernst & Whinney), Maria Pozzo (Aerospace Corp.), Peter Wild (Coopers & Lybrand).

[Ed. Note: The Applications Working Group chose to focus on Quality Metrics. Applications was considered to be too broad for meaningful discussion]

1. The objective of the working group was to deliver more specifically worded definitions applicable to measurement of data integrity (DI) in an applications systems and operating systems environment. This is a new field; the Orange Book concentrates on confidentiality and does not provide useful guidance.

2. The definitions of Quality Metrics could be either broadly set or technically focussed; a broader consideration was considered more appropriate at this stage. A basic evaluation criterion is the "level of trust". We did not see significant crosslinks with software quality assurance, but similarities with the quality control concerns of manufacturing.

3. The terms Quality, Integrity, and Goodness have to be carefully separated. Quality looks at individual properties, Integrity at the composite picture of various qualities. Goodness implies a desirably high level of quality or qualities, whereas Integrity is concerned with quality in line with expectations, which in itself could be for low quality.

4. *A priori* expectations must be decided and specified. Data Integrity would provide us with Reliance, Trust and Comfort; contributing to the value of the data. In that way, it is similar to the value of an audit of financial data, which gives us a limited written opinion with a lot of intangible value attached. Business must be sold on Data Integrity in order to attract attention and funding for measures to improve DI.

5. Ernst & Whinney surveyed users on their ranking of computer security components, and found them to judge availability as High ranking, integrity, defined as correctness and freedom from corruption, as Medium ranking, and confidentiality as Low.

6. Coming from a different angle to Quality Metrics, we could measure to what extent the integrity of a process and the resulting data is automatically enforced.

7. The group considered that the more general approach demands that integrity encompasses the whole business cycle, not just the computer aspects.

8. Integrity needs to address segregation of duties and other known control techniques. The Generally Accepted Accounting Principles (GAAP) are a general integrity policy. Such integrity policies statements must:

relate to the wide scope of the real world;

identify the boundaries of integrity;

define constrained data items, label them and preserve the integrity of the labels;

provide an audit trail on the processing steps;

distinguish between transient and persistent data items; and

put constraints on transactions by "forms".

9. Significant transactions need to be independently created and authorized. We need a statement of principles on "Well-formed Transactions" and "Transformation Processes".

10. The IVP (Independent Validation Process) is one tool to enhance DI. Its timing is important; a time stamp needs to signal at what time the validation was successfully completed.

11. The extent to which recoverability is part of DI was considered. Availability includes the responsibility for a coordinated or integrated recovery of transactions, issues of concern both to availability and integrity of the data.

12. As a summary, the group listed the following components of DI:

Accuracy
Pedigree / Audit Trail
Self-Consistency
Validity
Correspondence with the real world

13. One of the key aspects of DI is its concern with change. Both confidentiality and availability address an environment at a given, snapshot time. DI is interested in requiring change when it is appropriate, and stopping change when it is inappropriate.

14. DI is deeply concerned with controls over data, the processes, and the processing environment. Controls can be described as

preventive - stopping a DI violation from happening;

detective - recognizing a DI violation; or

corrective - repairing the damage done by a DI violation.

15. Another way of looking at dealing with DI violation distinguished six combinations of the three types of violations (Missing, Wrong or Added data) with their degree of impact (Known or Unknown).

3.1.5 Implementations/Models Working Group (Group 5)

Prepared by Stewart Kowalski

Willis Ware (Rand), Chair
Stewart Kowalski (Stockholm U.) Recorder
Marshall Abrams (Mitre) Sushil Jajodia (George Mason U.) Robert Jueneman (CSC) Milan Kuchta (Defence Comm. Est., Canada) Stanley Kurzban (IBM) Carl Landwehr (NRL) Stelio Thompson-Sittas (Expert Systems) Ravi Sandhu (Ohio State U.) Anne Todd (NIST)

The group tried as best it could to stick to the agenda laid out in the workshop outline which was (see Section 1.2):

I. Integrity Framework Elements

1. Introduction - Scope and Objectives
2. Concepts - Integrity, Quality , Value
   - Policy and Principles
   - Attributes of Quality
   - Roles of People
3. Basic Integrity Requirements
4. Quality Metrics

The first item of business on the agenda was an Integrity Framework.

FRAMEWORK

In the ideal situation, design and development should follow the general pattern of first using accepted principles to formulate policies which are then used to specify requirements that are implemented and tested.

In reality this design development cycle is generally not followed and often policy is used as a catalyst for the generation of principles. Thus, in the case of an integrity framework, policies have been roughly defined by the powers that be and now effort and research is being applied to establish principles that clarify and refine these policies.

Figure 3-2 shows the hierarchical framework mentioned above. Iteration is shown between policy and requirements and the design cycle is closed by having testing loop back to requirements.

It should be mentioned that in this framework, policy goals must be stated independently of products. This is shown in Figure 3-2 where implementation and test (i.e., product development and evaluation) are not directly looped back to principles but are fedback into requirements, thus averting a situation where products are directly connected to policy and policy is directly connected to products.

Another explanatory note to the framework is that it does not require that integrity be defined as absolute. That is, it should be possible to specify integrity requirements to the effect that 95% "integrity", (or whatever term one wants to use), will be obtainable with the design.

It should also be pointed out that this type of framework may be useful for a framework of security, and that a security framework and an integrity framework will often overlap one another.

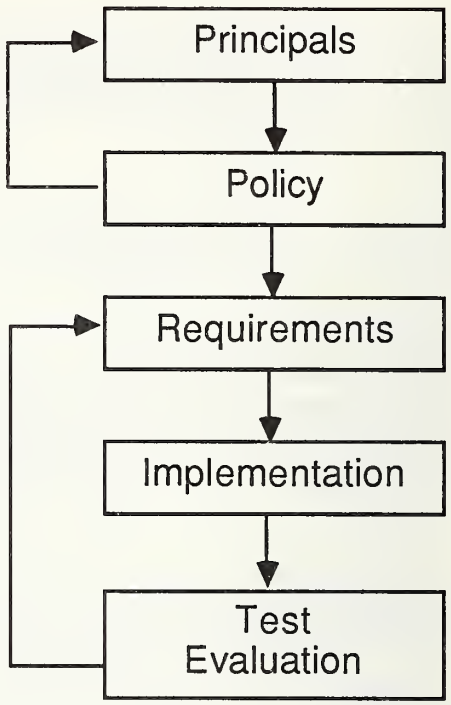


Figure 3-2: Framework For Integrity

SCOPE AND OBJECTIVES

It was agreed upon by the group that for the purpose of this workshop the discussion of integrity will be limited to data in EDP systems, including distributed EDP systems, but would not deal with the organizational data system, e.g, MIS. The definition of EDP systems and distributed EDP systems used is that defined by FIPS PUB 31 [FIPS31].

The group came to a stumbling block with the "a priori expectation" concept introduced by the workshop's strawman paper. It was the conclusion of the chairman that the concept "a priori expectation" was only confusing the discussion of integrity.

CONCEPTS

The concept or definition of integrity was more debated than discussed in the group. In the end, in order to resolve the debate and move on to other items on the agenda, a universal definition/formula for a property called P was agreed upon.

It was agreed that X can have some property P if it satisfies specification Y with confidence Z.

X has P

if

X Satisfies Y

with

Probability Z

P = {integrity, acceptability ..... etc}

X = {process, data, systems....etc}

Y = {accuracy, timeliness, completeness ...etc}

Z = (confidence, probability, discrete judgments....etc}

By not specifically stating what property P was, the group was then able to jump to agenda items of "attributes of quality".

THE ATTRIBUTES OF DATA QUALITY

As a starting point for the discussion, the attributes of quality presented in the morning session by Viiveke Fak were listed. The group then went through this list removing or adding items.

In general the group had great difficulty in coming to an agreement on a list of attributes of quality. Many of the attributes were interrelated and one attribute would be said to be a sub-attribute of another rather then a unique attribute of quality. For example, precision was said to be a sub-attribute of accuracy.

Since the discussion was slowly evolving into "how many angels can dance on the end of a pin" discussion, it was decided to use a real world situation of a patient's medical record to see what attribute of quality could be deduced.

At the end of the discussion a consensus was reached on six attributes for quality. The six attributes agreed upon were;

- Accuracy
  (e.g., spelling errors, numeric ranges)

- Timeliness
  (i.e., not stale for intended use, current)

- Consistency
  (i.e., intra data elements, intra and inter record)

- Completeness
  (i.e., all numeric places filled)

Attributes added to original list;

- Pedigree
  (i.e., where the data came from)

- Robustness
  (i.e, incorruptibility)

Attributes removed from the original list;

- Relevance

- Interpretation

Confidentiality was intentionally excluded from the list of data quality attributes and it was easily agreed upon by the group that it was an attribute of "process" quality.

PROCESS QUALITY

A brief discussion was held to deal with the problem of process quality. The following is a point summary.

- data only makes sense in some context and process is the means by which data is brought into context.

- a process is defined as the transformation of data from one quality domain(i.e, input data quality) to another (i.e., output data quality).

- in transforming the data from one quality domain to another, the quality of the data can either be preserved, increased, decreased, or destroyed.

- it was agreed upon that process quality has robustness and pedigree as attributes.

3.2 CONSENSUS VIEW FROM FRAMEWORK ELEMENTS DISCUSSIONS

As can be seen from reading the accounts of the Day 1 discussions (in Sec. 3.1) of the Integrity Framework Elements, consensus was limited. No agreement was reached on the framework elements per se. However, discussion among the group chairmen and workshop chairpersons led to the following consensus so that Day 2 discussions could progress.

3.2.1 Policy and Objectives for Quality

In order to achieve data quality, the following features need to be addressed by the systems in which the data are processed:

- Authorization
- Accountability
- Auditability
- Internal Consistency
- Separation of Duties
- "Real World" Correspondence
- Concepts of:
  - Constrained Data Items
  - Well-Formed Transactions
  - Recoverability
- Prevention of Missing/Wrong/Extra Data
- Change Occurs If and Only If Appropriate

3.2.2 Common Mechanisms for Quality

The following notions of common mechanisms for obtaining data quality in a system should be considered when data quality is an issue.

- Means of Attribute Measurement
- Trusted Transformation Processes
- Constrained Data Items
- Object/Subject Authorization
- Event Signalling
- TCB
- Application Specific Reference Monitors
- Separation of Duties
- Privilege Mechanisms
- Audit Trails, Logging Mechanisms

4. INTEGRITY IMPLEMENTATION REQUIREMENTS, APPROACHES, MODELS

4.1 OPERATING SYSTEMS AND SYSTEMS - GROUP 1 REPORT

Prepared by Tom Chen

Steve Lipner (DEC), Chair
Tom Berson (Anagram) Tom Chen (Wang) Ken Eggers (MITRE/McLean) Stu Katzke (NIST) Jan Kruys (NCR) Kurt Meiser (C&L) Tom Parker (ICL) Paul Peters (NCSC) Sig Porter (Consultant) Damian Saccocio (NRC) Steve Welke (IDA)

1. INTRODUCTION

The group's primary goal was to identify operating system features and mechanisms in support of integrity. The approach was to: (1) come to a reasonable agreement over a working definition of integrity properties, (2) develop a set of integrity control objectives, (3) characterize operating system features and mechanisms for meeting those objectives, (4) provide suggestions for initial guidance and criteria documents, and (5) recommend topics for research.

In general, [CLARK89] provides a reasonable definition of integrity that serves well as a framework for identifying integrity control objectives and corresponding operating system features and mechanisms.

1.1 BACKGROUND/DEFINITIONS

Although Bob Courtney has questioned how one can decide what to do about integrity before it is satisfactorily defined, the group sought to avoid getting immersed in a linguistic "wheel spin" over a definition of integrity. Regardless of what definition was assigned to integrity, it should still be possible to identify desirable operating system features in need of more attention. Nevertheless, the group did settle on a framework for integrity. In addition, we sought to avoid inconsistent use of the terms "mandatory and discretionary" and discussed the impact of integrity on the definition of a trusted computing base (TCB).

1.1.1 Integrity Framework

While the group agreed that the Courtney/Ware "a priori expectation" definition is a useful concept, we thought it was too broad an umbrella for integrity. For example, one could argue that the other two components of the "security triad" (i.e., confidentiality and availability) also fit under the Courtney/Ware umbrella. Consequently, we preferred to rename their "a priori" definition as "acceptability".

Essentially, we agreed with Clark/Wilson and focussed on those properties which give data and systems both internal consistency and proper correspondence to real-world expectations. (See Section III of [CLARK89])

1.1.2 Mandatory and Discretionary

The Orange Book rather narrowly implies mandatory as separation based on lattice oriented labels and discretionary as separation and sharing based on user identity and need-to-know. As with the WIPCIS at Bentley College, the group did not want to usurp natural language and agreed to define mandatory as allowing "no user choice" and discretionary as allowing "user choice".

1.1.3 TCB

The Orange Book defines a TCB as "The totality of protection mechanisms within a computer system -- including hardware, firmware, and software -- the combination of which is responsible for enforcing a security policy", and its rationale indicates that system elements excluded from the TCB need not be trusted to maintain protection. One viewpoint is that TCB elements fall into three categories: (1) elements that perform security functions (e.g., reference monitor, audit mechanisms), (2) elements that provide the TCB self-protection (i.e., protect itself from external interference or tampering), and (3) elements that do not fall into the previous categories but must still be trusted to perform correctly (i.e., perform their specified functions and nothing more).

When the security policy is restricted to unauthorized disclosure, it is often possible to demonstrate that certain system elements need not be trusted to maintain protection and fall outside of this "secrecy TCB" boundary. However, if the policy addresses unauthorized modification and/or other integrity properties, then the "integrity TCB" boundary may encompass the entire system. Given the notion of a secrecy TCB vs an integrity TCB, the distinction between the security kernel and the TCB is much wider and more important for integrity than for secrecy. While the security kernel can (and must) supply some of the protective functions and services, protective functions will undoubtedly also reside in the applications. Since the concept of an all encompassing integrity TCB is rather uninformative and unhelpful, it makes more sense to focus attention on the security kernel (i.e., the operating system mechanism that controls access, sequencing, etc.).

1.2 OBJECTIVES, SCOPE, AND GROUND RULES

The Orange Book's emphasis on preventing unauthorized data disclosure is reflected in its control objectives. Consequently, the group agreed to first define a set of integrity control objectives that were not addressed in the Orange Book Control Objectives. Given agreement on a set of integrity control objectives, we agreed to characterize corresponding operating system features and to make recommendations on guidelines and research topics.

The group felt that it should address integrity issues from the operating system's point of view (i.e., looking bottom up from the OS). Consequently, integrity issues addressed at the application level might not be visible or relevant to the underlying OS.

The group briefly discussed integrity threat models and essentially declined to thoroughly examine the issue, taking the view that threats addressed directly by the OS involved TCB self-protection (already addressed by the Orange Book). We felt that most integrity threats were specific application level issues and that the group's identification of integrity control objectives at the OS level should consider a more generic notion of integrity threats.

2. OPERATING SYSTEM CONTROL OBJECTIVES

The group identified integrity control objectives in two categories: necessary and optional. Integrity control objectives were categorized as necessary when they were clearly operating system responsibilities. The optional category encompasses those objectives which may not be compulsory at the operating system level. It is important to note that certain essential integrity control objectives could be appropriately addressed by the application and may not require operating system support. Consequently, the term "optional" applies specifically to whether it must be addressed as an operating system feature; and "optional" does not indicate that the objective is not essential to the system as a whole. In addition, desirable but unessential operating system control objectives are also categorized as optional.

2.1 NECESSARY OPERATING SYSTEM CONTROL OBJECTIVES

The following represents integrity control objectives which must be supported (i.e., necessary) by the operating system. The group admits that this first attempt may not be all-inclusive.

Protect Objects From Unauthorized Modification
Protect Programs From Unauthorized Execution
Binding of Data to Programs
Accountability
Enhanced User Authentication
Support for Application Level Audit Trails
System Level (i.e., privileged user) Separation of Duties
Assurance
Ease of Correct/Safe Use

2.1.1 Protect Objects From Unauthorized Modification

Protecting objects (i.e., data and programs) from unauthorized modification (e.g., write, delete, append) is a classic (and somewhat self-explanatory) integrity control objective in keeping with any reasonable view of integrity.

2.1.2 Protect Programs From Unauthorized Execution

Several group members were reluctant to specify Clark-Wilson "triples" as a necessary control objective and suggested a more generic approach that avoided explicitly requiring triples. In reality, this objective addresses the front half of the triple (e.g., control over which users can execute which programs). While programs must be protected from unauthorized

execution, explicitly requiring triples might be misinterpreted and reduce design and implementation flexibility at the operating system level.

2.1.3 Binding of Data to Programs

Binding data to programs requires that certain data can only be accessed via specified programs. This objective when combined with protecting programs from unauthorized execution addresses the Clark/Wilson triple. However, in general, with n_1 users and n_2 programs, using ACL doublets there are $n_1 + n_2$ access statements that can be made about a particular type of access for a particular object. With triples, there are $n_1 \times n_2$ statements.

2.1.4 Accountability

While accountability is an objective in its own right, it can also be considered as supporting other integrity control objectives. The following accountability objectives are both necessary for integrity and not clearly addressed by the Orange Book.

2.1.4.1 Enhanced User Authentication

While passwords may be adequate for some systems, they "should not be considered a realistic authentication method for a system with high expectations for data integrity" [CLARK89]. Consequently, systems should, at a minimum, provide appropriate "hooks" for external authentication devices.

It is also apparent that password mechanisms are not used to their full advantage. Additional guidance is needed to promote proper password management and use. (See [FIPS112].)

2.1.4.2 Support for Application Level Audit Trails

While the Orange Book identifies auditable events, it does not mention system support for audit records created at the application level. Audit records specific to an application and generated by the application program require operating system support for collection and protection.

2.1.4.3 Control of Privileged Users

Separation of duties in a manner consistent with the principle of least privilege for privileged system users (e.g., system administrators, security administrators, operators, auditors) is a necessary integrity control objective. However, any design/implementation must ensure that total elimination of a "super user/system god" does not lead to anomalous states in which no one can access security data or in some instances the system itself. Further research is needed to develop appropriate guidance.

2.1.5 Assurance

Assuring that a system behaves as advertised is certainly a necessary control objective already addressed in the Orange Book. Consequently, the group avoided any discussion of theorem provers and formal verification tools, particularly given an integrity policy's dependence on specific applications.

Assurance should also address ease of safe and correct use. Real world experience indicates that the systems with a high degree of security functionality and complexity (e.g., RACF/MVS) are often incorrectly installed and poorly maintained from a security perspective. While much research and work has been done on providing security functionality and mechanisms, methods to promote ease of proper installation, use, and maintenance require more research and attention. [PCIE88]

2.2 OPTIONAL CONTROL OBJECTIVES

The following integrity control objectives are categorized as optional at the operating system level because they could be performed at the application level and/or the group felt that they need not be explicitly required. For example, if a control objective is considered absolutely essential but could be properly addressed by the application, then the objective is categorized as optional for the operating system.

2.2.1 Bind Integrity Information to Objects

While the ability to bind integrity information (e.g., source, date, revision history, integrity status regarding IVP execution) may be essential to many environments, the binding could take place at the application level. For example, if the operating system supports binding of data to programs such that the data can only be accessed via a specific program, then the program could perform the integrity information binding.

2.2.2 Sequencing Support and Flow Control for Programs

Supporting separation of duties may require a sequence of programs (e.g., TPs and IVPs) to be executed in a particular order (e.g., Boebert-Kain assured pipelines). While sequencing support is highly desirable, it was not clearly identified as a necessary operating system integrity control objective. For example, all applications may not require sequencing, and it may be possible to support sequencing at the application level.

When the partition of change and separation of duty involve a sequence of programs (e.g., TPs) rather than a single program, the flow of data between the programs must be controlled by the application, the operating system, or both. For example, if a task is partitioned into a sequence of TPs with each TP isolated to its own domain, then information flow between domains must be properly controlled (e.g., Lee's partially trusted subjects [LEE88] or Boebert's domain definition and transition tables [BOEB85].

The group didn't have time to adequately address sequencing and flow control. Nevertheless, there was a sense that even if an application could provide the required controls,

the preferred approach may be to provide OS support. Sequencing and flow control clearly require further research.

2.2.3 Dynamic Separation of Duties

Clark & Wilson [CLARK89] indicate that separation of duty is achieved by requiring various TPs in a sequence to be performed by different people (i.e., the processes acting on their behalf). While static assignment of duties may be the simplest approach, it is "often necessary or desirable to reassign people to tasks dynamically" or "to keep track dynamically of the people who have executed the various TPs in sequence, and ensure, for any particular execution, that proper separation has occurred" [CLARK89].

3. CHARACTERIZATION OF OPERATING SYSTEM FEATURES

Meeting the identified integrity control objectives requires operating system support. This section discusses the philosophical points of view within the group, and identifies necessary and optional integrity features and mechanisms at the operating system level.

As with integrity control objectives, operating system features are characterized as necessary or optional. Integrity features are categorized as necessary when they are clearly operating system responsibilities. The optional category encompasses those features which may not be compulsory at the operating system level. It is important to note that an essential integrity feature could be appropriately addressed completely by the application without operating system support. In other cases, operating system support might be helpful to the application, but not required in that the application could support the feature itself. Consequently, the term "optional" applies specifically to the operating system and does not indicate that the feature is not essential to the system as a whole. In addition, optional can also categorize desirable but unessential operating system features.

3.1 POINTS OF VIEW

While technical purists might prefer to begin with a "blank sheet of paper" and assume the design of a system from scratch, the real world seldom allows such luxury. Nevertheless, it is a worthwhile exercise to consider a technically pure approach as well as real world pragmatism.

For example, several existing systems currently support enforcement of the Biba Integrity Model [BIBA77]. While the Biba model provides a "natural" policy for preventing unauthorized modification, it is not clear that the Biba model was ever intended to address partition of change and separation of duties. As Boebert and Kain indicated in [BOEB85], a system enforcing a Biba-like (i.e., partially-ordered lattice) integrity policy model to achieve separation of duties "either fails to enforce the desired restrictions or requires an exception from the policy at each step". On the other hand, Jeuneman [JEUN89] claims that these "criticisms are now generally recognized as being overstated" and that "the alleged difficulties go away immediately" when subjects are "trusted to operate within a range of integrity levels and more particularly integrity categories" (e.g., [LEE88]). While such an approach is feasible, the need for trusted subjects is unappealing.

Given the availability of Biba Integrity as a low level infrastructure in several existing operating systems, there is no question as to the pragmatic value of efforts such as [JEUN89], [LEE88], and [SHOC88] to enforce the Clark & Wilson model using Biba categories. While such approaches may be reasonably easy to manage, they require trusted processes which must violate the kernel's integrity policy, indicating that a Biba-like policy is inappropriate for the underlying security kernel.

It is clear from [BOEB85] and [SAND89] that strong typing and assured pipelines appear to be more elegant and better fitting approaches to Clark- Wilson separation of duties than a Biba-enforcing security kernel. Additionally, avoiding trusted subjects which must violate the policy enforced by the underlying operating system should allow a higher degree of assurance. Nevertheless, the existence of Biba enforcement in several existing systems (e.g., DEC, GEMSOS, ICL's VME) make it impractical to dismiss Biba solutions.

3.2 NECESSARY OPERATING SYSTEM FEATURES

Necessary operating system features include mandatory (i.e., no user choice) integrity controls, interface support for enhanced user authentication and application generated audit trails, and separation of duties and least privilege support for privileged users (e.g., system security officer and operators).

3.2.1 Mandatory Integrity Controls

In order to protect objects from unauthorized modification, the operating system must control the creation, modification, and deletion of data and programs.

In order to "ensure that data is modified only by selected programs" [CLARK89], the operating system must be able to bind data to programs. Appropriate integrity control also requires control over the user/processes allowed to execute programs. Binding data to programs and controlling execute access to programs can be combined to bind user, program, and data and enforce the Clark-Wilson access control triple.

It is clear that several systems are capable of accommodating the Clark-Wilson triple. In addition to the type manager/pipeline approach taken by LOCK and the Biba category approaches suggested by [LEE88] and [SHOC88], RACF and ACF2 also appear capable of supporting user/program/data bindings.

3.2.2 Interfaces to Authentication Devices

Users can easily and unilaterally invalidate their own identity by making their passwords easy to guess, by posting them, or by storing them in their own PCs. User education must make it clear that:

- They are responsible for actions authorized by the use of their passwords, and

- o This responsibility is consistent in nature with their other responsibilities to their employer or to national security, etc., (as the case may be).

In addition to user education, enhancements should make it more difficult for users to circumvent authentication mechanisms. This requires methods such as biometric devices and/or challenge-response tests involving a device (i.e., token) issued to a user that performs a cryptographic transformation on the challenge [CLARK89]. In addition, consideration should be given to developing an industry interface standard for external authentication devices.

3.2.3 System Interfaces for Application Audit

The operating system must provide a system interface for collecting audit records generated by an application program. In addition the operating system must provide appropriate storage media and access control protection for such audit records.

3.2.4 Roles/Structure for Privileged Users

Privileged users include those responsible for registering new users, managing and maintaining audit records, maintaining operating software, etc. Typical roles include System Manager, System Security Officer, Operator, and Auditor. The operating system must provide controls that separation of duty for privileged users is not circumvented. For example, the system programmers who develop software should not be permitted to install the software [CLARK89].

3.3 OPTIONAL OPERATING SYSTEM FEATURES

Optional operating system features include integrity tags and support for sequencing of operations and dynamic separation of duties.

3.3.1 Integrity Tags

Integrity controls will require binding integrity information to objects. For example, "the author of data should be recorded in an unforgeable way with the data itself", and since data "may undergo a number of modifications as it resides within the system, the record of authorship may need to be a record of the total change history of the object" [CLARK89].

3.3.2 Sequencing of Operations

Supporting the principle of least privilege and the separation of duties will often involve partitioning a task into a sequence of programs. Ideally, each program in the sequence would execute in its own domain with the operating system providing appropriate isolation. In addition, the operating system should provide a mechanism for passing data between domains (e.g., assured pipelines and domain tables [BOEB85], partially trusted subject [LEE88]).

3.3.3 Dynamic Separation of Duties

When a task is partitioned into a sequence of programs each in its own domain of execution, each domain should have a program being executed on behalf of a different user.

[CLARK89] outlines the need for the following support:

- Defining sequences and encoding allowed patterns of separation
- Logging record of each current execution of a sequence
- Require domains to be executed by different users and support dynamic access control
- Support access rights such as create, approve, review, and update

3.3.4 Enhanced User/Manager Interface

The degree to which sites incorrectly install, use, and maintain existing system security features affirms the need for enhanced user/manager interfaces. System security features and controls need more appropriate default settings, clearer structure, and user friendliness to facilitate proper installation, use, and maintenance. In particular, tools are needed to help a site ensure that their security controls remain in tact.

4. GUIDELINES

It is highly recommended that a set of integrity guidelines be developed that include amplification on operating system features in support of integrity, a description of best practices, and suggestions regarding the features required in various environments.

While establishing criteria is a worthwhile goal, integrity criteria must be preceded by appropriate demonstration and reference implementation. Additionally, it is recommended that the criteria separate functions and assurance.

Given the all encompassing nature of an integrity TCB, an evaluation process based on such a TCB was frightening to the vendors in the group. There was a particular concern over effects of the anomalies created by literal interpretations of integrity requirements which lose sight of the authors' original intent. If we ever get to the point of interpreting "Integrity Criteria", then evaluators must exercise extra care to avoid misusing the third TCB category to create bizarre and misguided literal interpretation anomalies. In general, while vendors are willing to respond to requirements, they have become skeptical of misinterpretation by the evaluation community.

5. SUGGESTED RESEARCH TOPICS

The group identified three major areas in need of further research:

- Separation of Duties for Privileged Users (see Section 3.2.4)
- Dynamic Separation of Duties (see Section 3.3.3)
- Enhanced User/Manager Interface to Facilitate Ease of Correct Use (see Section 3.3.4)

6. SUMMARY AND CONCLUSIONS

The operating system features identified by the group map well to the computer support described in Section IV of [CLARK89]. Minor differences arose from the groups operating system view point. For example, while Clark and Wilson call for support of their user/program/data triple, we felt that a more flexible and granular approach was appropriate at the operating system level and identified user/program and program/data binding as the necessary operating system features. Nevertheless, the group essentially agrees with Clark and Wilson.

While the group recommends developing integrity guidelines that provide further amplification of operating system support for the integrity features described in Section 3, we caution against issuing premature Evaluation Criteria prior to worked examples and the development of a referenced implementation.

Areas recommended for research are identified in Section 5.0

7. REFERENCES

BIBA77 Integrity Considerations for Secure Computer Systems, K.J. Biba, MITRE TR-3153, MITRE Corp., April 1977.

BOEB85 A Practical Alternative to Hierarchical Integrity Policies, W.E. Boebert and R Kain, Proceedings of the 8th National Computer Conference, Sept 1985.

CLARK89 Evolution of A Model for Computer Integrity", David Clark and David Wilson, NIST Invitational Workshop on Data Integrity, January 1989.

JEUN89 Integrity Controls for Military and Commercial Applications, II, R.R. Jeuneman, NIST Invitational Workshop on Data Integrity, January 1989.

LEE88 Using Mandatory Integrity to Enforce "Commercial" Security, T.M.P. Lee, Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 1988.

PCIE88 President's Council on Integrity and Efficiency, "Review of General Controls in Federal Computer Systems", 1988.

SAND89 "Terminology, Criteria, and System Architectures for Data Integrity", Ravi Sandhu, NIST Invitational Workshop on Data Integrity, January 1989.

SHOC88 Implementing Clark/Wilson Integrity Policy Using Current Technology, W.R. Shockley, Proceedings of the 11th National Computer Security Conference, October 1988.

4.2 TELECOMMUNICATIONS - GROUP 2 REPORT

Prepared by Ted Lee

David Clark (MIT), Chair
Ted Lee (TIS), Acting Chair
Eugen Bacic (Comm. Sec. Est.), Dennis Branstad (NIST), Viiveke Fak (Linkoping U.), Howard Johnson (Info. Intell. Scie. Inc.), Ted Humphreys (Brit. Telecom.), Stewart Lee (U. of Toronto), Rob Rosenthal (NIST), John Thurlow (Exxon), David Wilson (Ernst & Whinney)

1. INTRODUCTION

Unlike most other groups in this workshop, the Telecommunications group was able to decide fairly easily what the scope of its deliberations should cover and to conclude quickly that there are few technological obstacles to attaining arbitrarily high degrees of integrity in the handling of data within that scope. Accordingly, this report is primarily a summary of the state of the art and contains few calls for action, although it does note several institutional and political barriers to the widespread, economically feasible application of the relevant available technology.

2. SCOPE

The term "telecommunications" by itself means little more than "communicating over a distance." Neither a dictionary definition of the term nor any formal charge from the workshop organizers to our group gives the answers to questions like "communicating what?", "how fast?", "how far?", "between what or whom?", "over what medium?" or "in what form or representation?" As will be seen, even though the rest of the workshop clearly is intended to focus on matters having to do with the integrity of information processed by or stored in computers, the conclusions of our group do not depend strongly on whether computers are the endpoints of communication or not.

"Communications network", "communications system," "message handling system," and so forth mean different things to different people and all would seem to have something to do with the topic of this group. The main reason our group was able to reach its conclusions quickly was that it decided early on in its deliberations that however one might choose to define the scope of those terms, the only aspects of them that we need consider are those solely having to do with the transport of data. In short, we consciously excluded entities like "distributed processing systems" from our scope: such entities can always and should always, we contend, be analyzed as one or more dataprocessing systems interconnected by telecommunications systems, and it is the latter that we are addressing here. (One way to view our decision is this: "system" is a recursive term -- as everyone knows, any complex system can be decomposed into various subsystems, and those in turn into more subsystems and so on; for our purposes the recursion stops when each subsystem in the decomposition is entirely a telecommunications subsystem or entirely a data processing subsystem.) We recognize, as with other of our conclusions, that this is somewhat of an over-simplification: no matter how a

complex subsystem is decomposed there will almost always be parts that are "primarily" data processing but which do perform some communications functions, e.g., at the interfaces, and vice versa. Although it may sound contrived and with malice aforethought, since it coincides with the statement of integrity policy below in Section 3, our definition of a telecommunications subsystem is any subsystem whose primary purpose is to communicate information unchanged in any way from one point to another. Any subsystem that does something else, whatever it is, is not a telecommunications subsystem, although it may be decomposable into telecommunications parts and non-telecommunications parts.

(Note: we did not discuss the fine points of this definition much. Thus, for instance, one cannot conclude from this report, or from our deliberations, whether a module that converted between EBCDIC and ASCII or between an Autodin formatted message and an Internet SMTP message, should be regarded as part of telecommunications or as something else.)

3. STATEMENT OF INTEGRITY POLICY

In contrast with the other areas of this workshop the integrity policy desired for telecommunications is easy to state:

"That received data comes from a known class of source, unchanged (no additions, modifications, deletions, in original order, without repetition, etc.) with an acceptable degree of confidence."

There are three points to be noted about this statement of an integrity policy:

(1) It is recognized that there are applications where, although completely faithful ("unchanged") transmission of information is desired, it may be acceptable to have a slightly weaker integrity policy: that either data is received unchanged or if it is changed the fact that it has been changed is known with an acceptable degree of confidence. A system required to implement this policy rather than the previous one would have had different design choices made for it, but most of the discussion in this report would apply unchanged.

(2) The requirement that the "class of source" be knowable with acceptable confidence may seem at first both vague and unrelated to the central issue of data integrity. The point there is that no matter how accurately data is received from where it was originated, the use to which that data is put by its receiver may depend on what the receiver knows about the sender independent of anything the sender communicates about it (or him or her)self. Since only the telecommunications system "knows" who the sender (point of transmission) is in many cases, only the telecommunications system can tell the receiver something about the sender and thus the requirement to do so accurately becomes an integrity issue for a telecommunications system. The requirement is deliberately vague ("class of source") concerning what needs to be communicated about the source because that varies greatly from application to application: in some cases it is necessary to know the identity of the source (at some level of granularity --network, geographic location, system, person, etc.) whereas in others it is only necessary to know some property of the source (e.g., accreditation range of the system, evaluation class of the system) from which

one can decide how much to trust the information conveyed. In any case, it is generally a requirement of the telecommunications system to be able to supply appropriate information about the source of a given piece of data, (potentially as detailed as saying exactly who or what the source is) independent of anything the source does, with acceptable confidence that information is correct and has been correctly received.

(3) Narrowly speaking the injunction above about "acceptable degree of confidence" is not part of the integrity policy itself. However, it is not very practical to separate the requirement that information be communicated correctly from the requirement that one knows with an acceptable degree of assurance that it has in fact been communicated correctly. The ability of a telecommunications system to adhere to or support any of the integrity policies discussed above is vulnerable to attack from a variety of threats. For the most part the general types of threats and vulnerabilities are not different from those applicable to the enforcement of a confidentiality policy, although the detailed scenarios would of course be different. Following is a summary of the major threats; this list is intended to be comprehensive but it may have overlooked something.

4. THREATS AND VULNERABILITIES

The ability of a telecommunications system to adhere to or support any of the integrity policies discussed above is vulnerable to attack from a variety of threats. For the most part the general types of threats and vulnerabilities are not different from those applicable to the enforcement of a confidentiality policy, although the detailed scenarios would of course be different. Following is a summary of the major threats; this list is intended to be comprehensive but it may have overlooked something.

4.1 ACCIDENTAL THREATS

The integrity of information being communicated may be compromised or corrupted by any of several kinds of non-deliberate, accidental events. The major ones to be concerned about seem to be the following:

"laws of physics" -- imperfect communications media (noise), accidental interference from man-made or natural sources, hardware failures, race conditions, etc. can all cause information to be lost or altered.

"human operational error" -- at almost any point where a person is involved in the day-to-day operation or maintenance of a telecommunications system there is opportunity for human error to cause a loss of integrity of information.

"design and implementation error" -- even though a given system may be intended to have mechanisms such as those discussed below in Section 5 to protect itself against the above two kinds of accidental threats, the design and implementation of those mechanisms may have errors that either weaken the effectiveness of the mechanisms or themselves introduce a loss of integrity. Note also that even though the existence of a design or implementation error may be an accident, that error may be exploitable by the malicious attacks discussed below in Section 4.2.

4.2 MALICIOUS THREATS

Even though a given telecommunications system may be designed and implemented to cope well with the "laws of physics" or other forms of accidental error discussed above, the information it carries may be sufficiently valuable or important that it is in somebody's interest to attempt to deliberately cause the system to fail to preserve the integrity of the information it is communicating in one way or another. The particular kind of loss of integrity (false attributes of the source, altered information, loss of information, repeated transmission, completely bogus traffic, etc.) that would benefit an "enemy" most, or would be easiest to bring about, would vary from case to case. Following are the general kinds of malicious attacks a telecommunications system may be subject to:

"active wiretapping" -- whatever the particular communications medium may be, an attacker can in principle intercept the legitimate communications on the medium and replace them in whole or in part with false communications of his own devising, perhaps related to or derived from the legitimate communication, perhaps not; he may also just settle for intelligently or randomly adding noise to the communication, although that particular attack falls more into the denial of service realm. (An attack which simply over-rides the legitimate communication, e.g., through a stronger signal, without capturing and making use of the legitimate one also falls into the active-wiretapping category.)

"technical penetration of the interface" -- rather than direct attacks on the communications media, this encompasses any means by which the attacker exploits flaws in the communications system, its interface to an endpoint, or the endpoint itself to cause false information to be transmitted, for his benefit, from the sending end or delivered by the receiving end. An example here might be exploiting a weakness in the protection mechanisms of a host operating system to modify internal tables or even to directly call low-level communications routines and thereby to generate traffic that appears to come from someone else.

"subversion of system life-cycle" -- this includes all attempts to interfere with the development, distribution, maintenance, or operation of the communications system's hardware or software in such a way as to cause or allow, directly or indirectly, the intended integrity policy to be violated. System lifecycle subversion could be a means of inducing or planting a flaw (e.g., trojan horse software, faulty hardware) to be later exploited by a technical penetration of the interface. Note that this includes both cases where the "enemy" has direct access to some aspect of the life-cycle and cases where he doesn't but can somehow make someone who does have access become an unwitting agent (again, the trojan horse scenario comes to mind) to perform the subversion. Note that if portions of the system's software or critical tables are transmitted electronically, the integrity of that communications itself must be protected lest it be attacked by one of these means to effect a subversion of the system. (How the level of integrity required for the distribution mechanism for a system relates to the level of integrity it is trying to preserve is an interesting question without an easy answer but seems intuitively related to the question of what level of classification the TCB of a system handling classified information should be treated as to protect it from modification.)

5. INTEGRITY MECHANISMS

Well-understood technical and procedural mechanisms exist that can effectively counter all the threats discussed above. (Prevention of design and implementation errors is a separate topic, covered below in Section 6.) The primary ones are summarized here, along with which threats each is intended to deal with.

Error Correcting Codes --
Error Detecting Codes --
Protocols (error correction, sequencing, authentication, key management, etc.) --
The above mechanisms come in two forms: cryptographically strong and cryptographically weak. A cryptographically weak mechanism is one which is adequate to defend against accidental errors, such as noise in the communications medium, but which can be relatively easily circumvented by a conscious attack (such as modifying both a message and its checksum.) A cryptographically strong mechanism however is one which is computationally infeasible to circumvent, under the circumstances in which it is used, and thus serves to defeat active wiretapping. (In some sense the difference between a cryptographically weak or strong mechanism is in the eyes of the beholder; that topic is discussed below in Section 6.)

Reliable Media -- When physical and personnel security provide an adequate defense against an active attack, the threat of accidental errors may be reduced to an acceptable level merely by using a reliable medium (with perhaps only a simple error detecting/correcting code.) Fiber optics used for LAN's are an example of such a medium.

Trusted Systems Technology -- Many of the mechanisms used in a trusted computer system of class B2 or higher, implemented so as to assist in enforcing an integrity policy, are relevant to defending against a technical attack on the interface, although in many cases all that may be needed is simple domain isolation. (The issue of correctly coupling the information protected by an integrity-policy-enforcing TCB to the communications system is a separate one and is discussed below in Section 8.)

Physical and Personnel Security -- These mechanisms are the primary means of defending against subversion of the system life-cycle, although they can also prevent active attack (by using a protected medium) and technical penetration of the interface (by preventing an "enemy" from having any access at all to either end of the telecommunications system.)

6. ASSURANCE MEASURES

The primary technical protection mechanisms listed above are all vulnerable to exploitation of design and implementation errors. Various measures, more-or-less well-understood, are available to assure that the mechanisms in a given communication system have in fact been designed and implemented properly. Without use of these measures, especially in a high-threat environment, the inclusion of particular integrity mechanisms can only give the illusion of adequate protection (as demonstrated by the ease with which amateur cryptographic al-

gorithms have been broken or the ease with which operating systems not built to TCSEC standards have been penetrated.) Three particular sets of measures are applicable:

Cryptographic Certification -- Determining that a particular error-detecting or correcting coding algorithm and its related protocols (including crypto key management) is cryptographically strong enough for a given application involving an active wire-tapping threat is presumably well- understood by the national cryptographic community, but it is a technology that, unlike anything else discussed here, is not publicly knowable. However, the results of the technology are available in that particular algorithms are approved for particular purposes and can be confidently used in systems without knowing the approval process or even the algorithm itself; the only concern might be that the use of cryptographically-strong algorithms for integrity protection (as opposed to confidentiality protection) is a relatively recent idea and thus may not be as well-understood, even by the cryptographic community, as the older confidentiality-prevention technology is. Ameliorating this concern is the observation that active wire-tapping to compromise integrity must almost always be in real-time whereas the need to decrypt intercepted material from a passive wire-tap in order to breach confidentiality is hardly ever so urgent.

Protocol Verification -- Although somewhat in its infancy, the technology to demonstrate through formal proofs that a given suite of protocols is "correct" has been successfully applied to a number of examples. The more complex the interactions involved in sequencing, authentication, authorization, and cryptographic key management, the more use of such approaches becomes attractive.

Trusted System Assurance Measures -- Any place where software and hardware protection measures are used to separate trusted parts of a communications system from untrusted parts and any other places where they must be relied upon to operate correctly in order that integrity be preserved are candidates for the application of trusted system assurance measures from classes B2 through A1 of the TCSEC.

7. NEED FOR GUIDANCE

One difference between a communications system concerned about preserving the integrity of information and one only concerned about preserving confidentiality is that other than the long well-understood principles for engineering communications systems with specified (acceptably low) error rates, there does not exist any codified guidance that helps one determine whether a particular such system is "good enough" for a particular application, particularly in the face of an active threat. It is assumed this same observation will be made and discussed more at length by the operating systems group since it applies even more so there. In short, the problem is that there is no analogue of the TCSEC Environments Guidelines (Yellow Book) [ENVIR85] for integrity policies whereby one could say something like "given a system handling information requiring protection at integrity level x, accessed by people trusted at level y, exposed to threat types z and w, a system of evaluation class B2 would provide adequate protection" nor is there an extension of the features and assurances in the various evaluation classes that covers the kind of error detecting and correcting codes, protocols, and crypto algorithms relevant here. (It can even be argued that without even going so far as trying to decide whether a particular system is "good enough" for a given operational

environment we do not have any well-codified way of specifying the properties of an environment that are relevant to the decision: in the realm of preserving the confidentiality of information the concepts of clearance and classification -- even outside the National Security Establishment -- are useful ways of characterizing how severe an operational environment is in terms of the threats present and how important it is to defend against them; no commonly-accepted analogous concepts are yet available in the integrity realm.)

8. LOOSE ENDS

Time did not permit our working group to cover, even briefly, a number of other relevant topics (even within the narrow scope that we defined telecommunications to encompass.) Without much comment, then, here are four in particular that should be explored before one could regard the subject as being close to complete:

Oversimplification -- We recognize that in summarily defining telecommunications to include only the transport of information, unchanged, between two endpoints we are both being imprecise and oversimplifying the problem. Some of that was noted above (is a format/character set conversion routine part of the telecommunications system or not?); the consequences of having such a narrow definition of scope should be re-examined. (Even though the operating systems group has implicitly embraced "distributed systems" as in its scope, we must make sure that the definitions of scope of all the groups do in fact cover all relevant areas.)

Composition/Interface Incompatibilities -- Even though a given telecommunications system may be designed to adequately preserve the integrity of information in the face of the threats it is exposed to when considered as an isolated system, the consequences of coupling that telecommunications system into the systems at either end that will be using it are not obvious. The endpoint systems may have certain expectations about the information to be carried (e.g., integrity labels supplied by a trusted part of the end system) that a routine, even if of high-integrity, communications system design may not anticipate.

Unresolvable trade-offs -- There are always a number of competing goals for a system; in this area the ones that come to mind are integrity, confidentiality, availability, bandwidth, delay, and economics. It is not clear that these can be arbitrarily specified. We have a suspicion, for instance, that high levels of integrity, which require high amounts of redundancy in a noisy medium, are inherently in conflict with high levels of confidentiality since redundancy is known to aid in breaking a cryptosystem. The limits here need to be explored.

Relevance of the TNI -- We recognize that the Trusted Network Interpretation (TNI) [TNI87] has something to say on this topic, especially in Part II. Whether it is of much utility or applicability needs to be examined further.

9. NON-TECHNICAL BARRIERS

As already noted, we believe that the technology to provide integrity over a telecommunications system adequate for any practical application even under an extremely hostile

threat environment is readily at hand. However, we see a number of non-technical barriers to the widespread and economical use of that technology:

Agreement on Interfaces and Protocols -- Although much progress has been made in specifying various protocols suites and internal operating system interfaces, e.g., under the ISO framework or in communications and networking extensions to Unix(tm), it cannot be said that there currently exists a widely-accepted and implemented set of them that completely covers the variety of integrity-related service parameters that fall under the responsibility of a telecommunications system.

Adequacy Guidance -- As noted above in Section 7, there does not exist either the technical basis for deciding "what is good enough" for a telecommunications system in terms of integrity mechanisms and assurance measures nor, as a consequence, any procedures, doctrine, or standards institutionalizing that basis. Note, as also discussed earlier, that part of the reason for this lack is that apart from engineering specifications of an acceptable error rate in a noisy channel there is no common way of characterizing "how much" integrity protection is needed for a given class of data or how great the threat it is subjected to in a given environment, i.e., the common vocabulary and succinct form of characterizing a confidentiality-preserving environment in terms of clearances and classifications (or analogous concepts) does not exist here.

Certification or Evaluation of Products -- Similarly, there is lacking both a metric for evaluating telecommunications products providing integrity protection and any institutional way of certifying a given product. Although the TNI [TNI87] provides a start by providing a partial framework for characterizing a product it does not, as yet, provide a rating that serves for telecommunications systems the same function that a DoD Criteria [TCSEC] rating provides (regardless of whether confidentiality or integrity is of concern) for an operating system.

Export Problems -- Cryptographically-strong integrity mechanisms of necessity involve cryptographic algorithms. The exportability of those algorithms or of equipment using them is undecided, although there may be precedents in that an algorithm that cannot be routinely exported if it is used for confidentiality protection (e.g., DES) can be exported if it is only used for authentication purposes.

Transborder Data Flow -- Laws in various jurisdictions require that some kinds of data which flows over the border must not be encrypted; how those laws apply to data which is encrypted for integrity protection is not known, although it is noted that a copy of such data can always be also transmitted unencrypted along with the encrypted data without detracting from the telecommunication system's ability to preserve its integrity.

Cost/Performance -- Even though the technology to provide very high levels of integrity in a communications system is in principle fairly straight-forward, providing that technology at an acceptable cost in a system of a given communications rate may depend on economies of scale that can only be achieved if a wide-spread market exists, i.e., if many of the non-technical problems presented above are resolved.

10. CONCLUSIONS

The conclusion of the telecommunications working group of the NIST Data Integrity Workshop is this: protection of the integrity of information handled by a telecommunications system can be provided to essentially arbitrarily high levels of confidence without the need for any major technological advances. There are, however, a number of institutional and cultural barriers to the adoption and wide-spread exploitation of that technology that must be addressed.

4.3 SYSTEM SERVICES - GROUP 3 REPORT

Prepared by Grant Wagner

Grant M. Wagner (NCSC) Chair
Deborah J. Bodeau (Mitre) David A. Bonyun (AIT) Rae K. Burns (Kanne Assoc.) Terry Mayfield (IDA) Roger L. Miller (IBM) Lee Ohringer (Dept of Treas.) Carl Pabst (Touche Ross Int'l) Sylvan Pinsky (NCSC) David Rosenthal (Odyssey Res. Assoc.) Marvin Schaefer (TIS) William R. Shockley (Gemini Comp. Inc.) Gary Smith (George Mason U.)

1. SCOPE

System services refers to all of the services provided by a system to its users (e.g., data base support, electronic mail, office automation, transaction processing support, etc.). In this report we will focus upon data integrity issues as they relate to Data Base Management Systems (DBMSs)[1]. We make no claim that our discussion of DBMSs will or will not apply to other system services. We simply chose to address DBMS issues because we felt that the entire area of system services was too broad to address in the time available and the majority of group members had done previous work in DBMS controls of one form or another.

This report will not address data integrity controls that are implemented as part of a particular DBMS application. This report will instead focus on what controls can be implemented in the underlying DBMS in hopes of reducing the burden of application dependent TP certifications.

2. DBMS INTEGRITY CONTROLS AND CLARK & WILSON

We chose to look at existing DBMS integrity controls and see how they could be used to implement the Clark & Wilson notions of Independent Verification Procedure (IVP)[2], Transformation Procedure (TP), Constrained Data Item (CDI), access triple and separation of roles.

2.1 UPDATES AND QUERIES

Since DBMSs allow both updates and queries we decided that we needed to consider both. We decided that the Clark & Wilson notions could be applied to query-based situations but that the effect would be to provide some form of inference control. We did not pursue this further as we felt this was a confidentiality concern and thus outside the scope of the effort. This allowed us to focus on DBMS operations that were primarily updates.

---

[1] In focusing upon DBMSs, this report will deal only with "traditional" DBMSs (e.g., we will not discuss the special problems of distributed or multi-processing DBMSs).

[2] Note that different authors use different words to explain the acronym IVP, e.g., "I" can mean Integrity, Initial, or Independent while "V" can mean Verification or Validation. This usage needs to be standardized.

2.2 THE ACCESS TRIPLE

Most available DBMSs provide only the capability to specify two component access controls, (e.g., <user, data>, <user, transaction>, or <transaction, data>) whereas the Clark & Wilson model uses access triples. The group was divided on whether or not one should be able to construct appropriate groupings of these access doubles such that the function of the access triple was provided. Unfortunately, no consensus could be reached on this topic. There was agreement that the "precompiled transaction"or "canned query" capability in some current DBMSs could serve as triples.

2.3 TRANSFORMATION PROCEDURES

The group quickly agreed that the DBMS must have some capability to bind names to TPs. This is required to allow TPs to be referenced by access tuples and other TPs.

The group also agreed that the DBMS must have the capability to encapsulate CDIs such that they are only manipulated by the appropriate TPs. This raised the question of what happens when two or more TPs designed for different missions (e.g.,payroll and medical services) manipulate the same CDI (e.g., employee sickleave). The group also described the problem as multiple "paths"to the same data. If TPs are viewed as independent, when they in fact reference the same data, data integrity problems will likely arise that can not be addressed by independent TP certification. This did not mean that only one TP should be permitted to manipulate a CDI. However, if more than one TP does manipulate any part of a CDI, then the certification of all TPs which operate on the same CDI must be done in the context of all of the other TPs which manipulate the same CDI. This led some members to the conclusion that one may need to expand the triple to include the context of the certification (i.e., what the TP is certified to do or not do).

2.4 INDEPENDENT VERIFICATION PROCEDURES

The notion of TPs in a DBMS environment implies that the user interface is constrained to a subset of the entire DBMS interface. In fact, the user will be constrained to the set of operations on CDIs that were anticipated by the designers of the TPs. When used in real life applications, in most cases this will not be unreasonable. However things happen in real life that are not, and frequently cannot be, anticipated by the designers of the TPs. This means that any real life application will have an uncertified "override" capability that will allow some user to manipulate CDIs in ways not specified by any TP. Thus, in order to restore data integrity, the DBMS must provide some support for IVPs. Within the limited time provided, the group did not identify or propose any DBMS-based mechanisms to support IVPs. In fact, all of the suggested scenarios required some sort of reliable human reporting that validated that the override maintained data integrity. If the override did not maintain data integrity, the human was required to restore data integrity.

2.5 SEPARATION OF ROLES

The group addressed separation of roles by examining how one might implement a DBMS application to issue checks. (The example can be found in Section 5.) This example showed

that some "dynamic" separation of roles constraints can be implemented with mechanisms (e.g., data definition languages, data consistency constraints) that are commonly available in today's DBMSs. However, trying to implement even moderately complex constraints in today's DBMSs would be awkward and clumsy, probably resulting in a system that would be hard to maintain and certify.

3. SOME THOUGHTS ON CONFIDENTIALITY AND INTEGRITY

The group also considered how confidentiality and data integrity interact. It has been said that integrity and confidentiality work against each other and that one must always choose which is more important. We believe that this is too strong. We finally agreed that frequently integrity and confidentiality are mutually supportive and in fact often depend upon the same basic mechanisms. We also agreed that there are circumstances (e.g., where integrity constraints "cross" mandatory access control levels and vice versa) where integrity and confidentiality must be traded one for the other. Under these circumstances one must look for engineering compromises that will allow the proper balance needed for the desired application. The group observed that one must begin seeking the desired balance very early in the design of the system because early design and development decisions that ignore either confidentiality or integrity will limit (or completely preclude) opportunities to provide the needed protection during later development.

4. REFLECTIONS/RECOMMENDATIONS

While the group was able to use existing DBMS features to implement separation of roles controls, we were, however, unable to use existing features in a way that would support easy maintenance and certification. We recommend that data definition and/or consistency check features be enhanced to provide operators that lend themselves to the expression of integrity controls and to allow separation of integrity controls and traditional data.

More research and worked examples in real life applications where both integrity and confidentiality are necessary are needed to expand the currently limited set of engineering compromises available to DBMS developers.

5. EXAMPLE - GENERATING CHECKS

This section describes an example of how we can achieve dynamic separation of roles using what exists today at the applications level. We believe that this could be better performed at the DBMS level, provided the proper tools were made available. For this example, we assumed two data types, a check and a check register. We want to ensure that:

1. every check is requested, approved, and signed before issue
2. no single user does more then one of the above for any given check
3. no user participates in the processing of a check for themselves

We defined a check to have the following data fields: date, check number, payee, amount, signature. We defined the check register to have the following data fields: check number,

requestor's name, approver's name, and signatory's name. We defined four TPs: request_check, approve_check, sign_check, and issue_check.

Using commonly available data consistency features we defined some rules that would insure that our three constraints above were always met.

1. A check cannot be issued unless all fields have a valid value.
2. Requestor's name cannot equal Approver's name.
3. Requestor's name cannot equal Signatory's name.
4. Approver's name cannot equal Signatory's name.
5. Requestor's name cannot equal payee.
6. Approver's name cannot equal payee.
7. Signatory's name cannot equal payee.

Rules 2 - 7 are simple and these types of rules can be enforced by the DBMS provided the DBMS provides the Data Base Administrator with the ability to define limits on the data values placed in particular fields. Fortunately, many available DBMSs provide such a capability in what they call data definition languages or consistency constraints. This means that only the DBMS and the expression of the rules need be certified. Rule 1, on the other hand, would likely require some of its enforcement in the issue_ check TP thus additionally forcing this TP to undergo a certification.

This example did require us to introduce additional "data" fields (e.g., Approver's name) into the database for the sole reason of providing separation of roles. We also added a number of "consistency" constraints solely for separation of roles purposes. This use of additional data and constraints allowed us to achieve our goal at the expense of mixing "real data" (e.g., amount, payee) with previously unnecessary "control data" (e.g., Approver's name). Existing DBMS languages do not provide an elegant or clean way of separating this "control data" from the "real data", thus complicating maintenance and certification.

4.4 APPLICATIONS - GROUP 4 REPORT

Prepared by William H. Murray

Chair: William H. Murray (Ernst & Whinney)
Group Members: Robert Baldwin (Tandem Comp.) Joseph Beckman (NCSC) Nander Brown (SBA) Peter Capek (IBM) Karl Krueger (World Bk) Dale Miller (Irongate) Maria Pozzo (Aerospace Corp.) Dennis Steinauer (NIST) Anne Todd (NIST) Douglas Varney (Kans. St. U.) Peter Wild (Coopers & Lybrand)

1. CHARGE

This group was asked to look at the application implications of the requirement for data integrity. We were specifically asked to include both structured and unstructured applications.

We tried to keep a number of examples in mind during our deliberations. These included:

* games
* missile launch and targeting
* aircraft maintenance scheduling
* business applications (e.g., payroll, general ledger, receivables, etc.)
* strategic applications (e.g., reservation systems)
* medical diagnosis
* document retrieval
* business application program development
* other

1.1 BOUNDARY ASSUMPTIONS

We did, however, make some boundary assumptions that excluded some computer uses. We focused on multi-user systems to the exclusion of single-application-only machines (e.g., arcade games, automatic teller machines, etc.). We also excluded untrusted single-user machines, and applications where only the owner of the application would rely upon the results. (David Wilson encouraged us to include this case, but upon reflection we concluded that the only way to do so would be to push the system into the trusted system case.)

In considering the following cases:

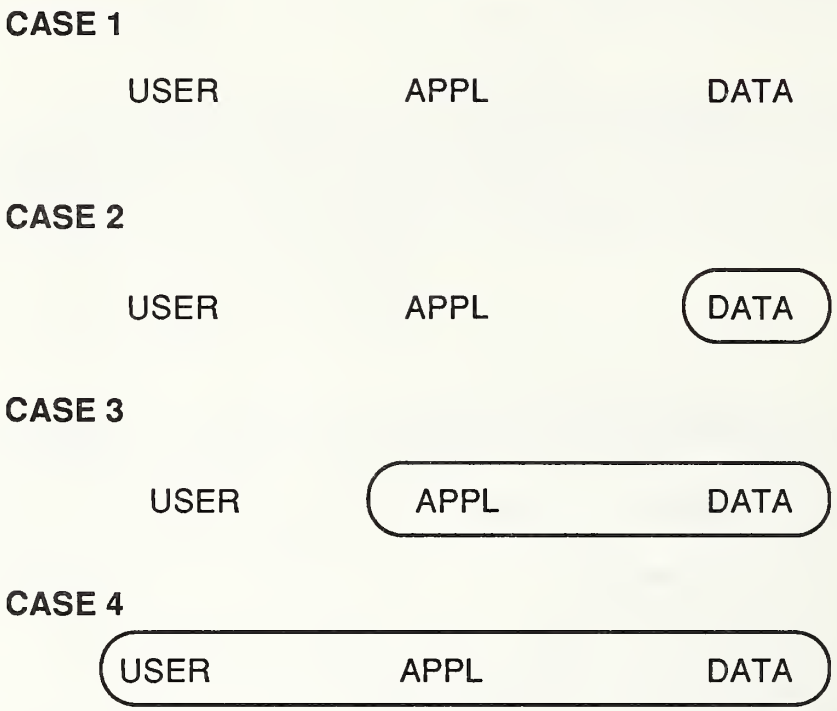


Figure 4.4-1 Boundary of Trust

where the line represents the boundary of trust, it is Case 3 that is the interesting case. We assert that in Case 1 nothing can be said about integrity. In Case 2 it would be required to restrict all write access. Case 4 does not represent a problem.

The group concluded that there must be mediation between the user and the data. It is equally clear that access between the user and the application must be mediated. C-W requires that the TP mediate the user's access to the data ("Access to CDIs must be via TPs). It is not clear how much of this mediation can be generalized across TPs and applications. There was some discussion of an application reference monitor, but it was never clear whether this function would be local to the TP, the application, or the system.

The group considered the level of access that the application had to the data. For example:

* physical
* device or device driver
* access method or data set

* file server
* structured database
* type manager
* combination of the above

While we felt that it would be easier to make statements about integrity in the presence of only one of these and that it would be easiest with type managers, we recognized that many applications would require or reserve low level access.

Finally, we distinguished between applications and tools. We did not consider editors, formatters, spreadsheet, database managers, etc., independent of their use, were applications. We concluded that word processing technology applied to inter-office correspondence was the same application as the tool used to prepare contracts in a law office.

2. PRINCIPLES AND CONSTRUCTS

The group identified three principles that were important to our work.

2.1 LEAST PRIVILEGE

The group felt that the principle of least privilege is necessary to be able to make statements about the integrity of data. C-W suggests this. The group believes that the principle must be applied at every layer of the system. For example:

* A user should be restricted to necessary applications and TPs (e.g. transaction types)
* TPs should be restricted to only the CDIs required
* the combination of the user and TP might be restricted to only part of a CDI.

2.2 SEPARATION OF DUTIES/SEGREGATION OF FUNCTIONS

2.2.1 Types of Duties/Functions

The group elaborated some of the kinds of separation of duties and segregations of functions that may apply to an application. These include:

2.2.1.1 Origination from Approval

This rule states that sensitive transactions must be independently originated and approved. It is most often seen in business applications but it is also seen in business application program development where it requires that changes to programs be approved by management.

2.2.1.2 Custody from Control Data

This rule requires that access to a resource be divided from access to the data used to control the resource. This is the rule that says that he who has physical custody of the inventory must not have write access to the inventory control system. It applies even when the resource itself is data.

2.2.1.3 Creation from Maintenance

This rule suggests that it is useful to separate the creation and naming of a record or other data object from maintenance or update of the object. This is the rule that says that he who creates the ledger record, i.e., creates vendor records, must not be able to process transactions against that ledger, i.e., approve invoices for payment. It also suggests that original development of a computer program should be separate from the maintenance of the program.

2.2.1.4 Processing from Rules

This rule says that he who can change the rules under which data is processed, i.e., modify the program, should not be able to process the data, i.e., execute the program. It suggests that programming should be separated from execution. (The exception to this is the case in which only one person will rely upon the results as in personal computing.)

2.2.1.5 Static and Dynamic Separation of Duties

The group assumed two kinds of separation of duties, i.e., "static" and "dynamic." The rule that requires separate origination and approval can be used to illustrate both kinds. Static separation occurs when duties are divided along the lines of transaction type, for example, when user A can do only originations and user B only approvals. Dynamic separation of duties occurs among instances of transactions within a type, as for example when user A can either orignate or approve as long as it is for different instances.

Dynamic may also include separation based upon the content or the context transaction. For example, user A might be able to unilaterally approve a transaction for under $1000 but require a counter for larger transactions. User B might be limited to employees or customers assigned to his branch.

Historically, systems have often supported division along the lines of transaction type. However, such separation along the lines of instance, content, or context that have been supported, have been supported only by applications.

2.3 THE WELL FORMED TRANSACTION

The group acknowledges its debt to C-W for the concept of the well formed transaction. However, we elaborated the concept to identify some of its characteristics.

2.3.1 Limited in Scope

It is equally indebted to the Working Group on Granularity of the first WIPCIS for its discussion of the scope of TPs. It agrees with that group that the TP must be limited in scope.

However, we assert that while it is necessary that the TP be limited in scope, it is not sufficient. It must also be simple, i.e., it must be of limited complexity or simply composed of things that are.

2.3.2 Obvious

The objective of the first two rules is that the TP, and ultimately the application, be obvious as to their intent.

2.3.3 Whole

Likewise the TP and application must be complete and sound. Only when these conditions are met can we expect to have data integrity.

While computer applications have seldom conformed to all of these principles, we have seen sufficient examples to convince us that they can.

3. CONCLUSIONS

3.1 INTEGRITY PHILOSOPHY

The group asserts that it must be possible for management to prevent, attribute, constrain, and partition change in a manner consistent with the requirements of the application.

3.2 INTEGRITY POLICY

The group believes that integrity policy is likely application dependent. We were unable to identify any policy statement that seemed to us to be both specific enough to be useful and general enough to embrace the set of applications that we had under consideration.

3.3 DATA INTEGRITY vs. SYSTEM INTEGRITY

The group agreed with the premise of the workshop that data integrity has not received the same attention as confidentiality. While focusing on data integrity, the group concluded that system integrity, controllability, and auditability are all aspects of a single property. Each is essential to the maintenance of the others.

3.4 APPLICATION KNOWLEDGE

The group concluded that the application had to be accountable for the integrity of its data. Further, that in order to be effective, that the application would have to have knowledge of the environment. We used the term "integrity perimeter," and concluded that the boundaries of such a domain had to be decidable and known to the application.

3.5 LOCALIZATION

We concluded that access to data needed to be localized both inside and outside the application. That is, not only should control of data be localized within the system, for example in a database manager, but also within the application, that is, within one or two modules.

3.6 PARTITION AND CONTENTION

The group concluded that the principle new requirement is the institutionalization of partitioning and contention within the system. This includes the concept of "strong separation of duties" as espoused by Parker and C-W.

3.7 NO "ALL PRIVILEGED" USERS

The group called for an end to the current practice of designing systems in such a way that there is an "all privileged" user. Whether called the superuser or system manager, the practice of having such a user is inconsistent with the objective of data integrity. Many systems not only assume the existence of such an individual but even make it mandatory.

The group recognized that there are cases in which the existence of such a role is appropriate, as in a small sole proprietorship. They also recognized that the absence of such a user might, in some instances, make corrective action more difficult. In the absence of very good design, it might even lead to a "horn-lock" or "deadly embrace" in which control of the system could be lost. However, we have seen sufficient instances of such design to convince us that the current practice is a designer's crutch.

3.8 ADDITIONAL ABSTRACTIONS

The group identified the need for additional layers of abstraction between that of application and that of transaction type. First, it seemed to us that application and transaction type are too far apart to support many of the administrative and separation requirements. We also noted that many systems and applications support grouping of users or resources along such lines as category, department, project. However, most of these appear to be ad hoc rather than ordered and general. The group feels that useful work can be done in this area.

4. REQUIREMENTS

The working group is indebted to Robert Jeuneman for the identification of the requirements. Our list is an elaboration on his.

4.1 IMMUTABILITY

As suggested in our conclusions, as well as by Biba, C-W, and Shockley, the basic requirement for data integrity is control of change. "Integrity is preserved when only appropriate, intended, and authorized changes take place."

The working group felt that there are a number of mechanisms available to resist changes to data. These include non-eraseable media, environmental controls (including binding and access control), digital signatures, and typed data objects. The latter two are of particular interest. While they are sparsely used to date, they are both flexible and broadly applicable.

4.2 ATTRIBUTION

Likewise, a key to directing and restraining change is the ability to fix accountability for all significant and material change to a single individual. As suggested by C-W, among others, it must be possible to fix accountability to the individual in an unforgeable manner. While passwords have worked well for some applications and environments, they are clearly limited. Biometrics can be both effective and efficient for applications where the terminals are trusted, clustered and shared across users. One-time passwords are preferred where terminals are not trusted and where the user may appear at many points.

The group questioned whether the application had to authenticate the user or could rely on some other process in the environment. The closer the authentication process is to the user, the more rational it will appear to him, but the less trusted it may be to the application. The closer it is to the application, the more trusted, but the more different processes the user must be known to. We concluded that the application should employ the process closest to the user in which it has sufficient trust. As a normal rule the application will have some trust in its environment; at a minimum it must rely upon the environment to protect it from interference and contamination. However, when it has no trust in any process outside itself, then one-time passwords are the only workable choice (since biometrics require both a trusted reader and path).

4.3 LOGGING AND JOURNALING

The group resisted the use of the term "auditing" and preferred to have separate requirements for logging, journaling, reporting and signalling. They stressed that these should be sufficient in number, independence, and immutability to be trustworthy. With Jueneman, the discussion group believes that digital signatures can be very useful here.

4.4 REPORTING AND SIGNALLING

Reporting and signalling are the second parts of the "audit" requirement. They provide management with sufficient visibility into the behavior, use and content of the system to enable them to detect variances from the expected and take timely corrective action.

4.5 PEDIGREE/PROVENANCE

However, the group felt very strongly that while logs and journals are necessary for accountability, they are not sufficient. Neither are they sufficient for proper partitioning. They felt that the data's pedigree and provenance must be immutably bound to it. [Provenance, as used here, is defined as the history and record of custody of an artifact that attests to its membership in a class, its unique identity, or its authenticity. For a data object this includes

its author, change history, sources and procedures employed in its preparation, authorizations, approvals, and other vouchers.]

While much of this data has been recorded in logs and journals, for reasons of economy of storage, it has usually been kept in archival storage. In order for it to be available for the purpose of enforcing partitioning it must be bound to the data object and kept in the same store.

4.6 VALIDATION

Validation, as described by C-W and Jeuneman, appears to be highly application dependent.

4.7 PARTITION/COMPARTMENTATION

The group concluded that the most significant new requirement identified by focusing on the data integrity requirement is for more granular partitioning and compartmentation. This includes the ability to separate duties along the lines of role (we define role to be an exclusive set of privileges associated with a particular duty; the privileges associated with a role should not be combined with other privileges), department, project, level of management, transaction type, transaction instance or other useful lines.

While we agreed with C-W that the decision as to how TPs should be separated is a decision that must be made external to the system, we felt that the enforcement of these rules at authorization time must be made internal to the system.

4.8 IDENTIFICATION

We agreed with Jueneman that the enforcement of unique identifiers must be done by the environment, rather than the application itself, in order to insure their exclusiveness over the widest possible domain. We also agreed that the system was in the best position to provide confidence in the name, particularly across applications or environments.

4.9 AUTHORIZATION

The group concluded that the authorization function needs to be more responsive to the requirements of applications. A large percentage of problems result from the rules themselves. Problems in the rules often result from the knowledge requirement imposed upon the authorizer in order to map application abstractions onto system controlled objects.

4.10 TRUSTED DATE AND TIME

The group agreed with Jueneman about the requirement for a trusted time-stamp process. We do not underestimate the difficulty of providing such a function but believe that a large part of the requirement can be met with a mechanism as simple as the system clock.

4.11 SYNCHRONIZATION AND ORDERING

This requirement is an elaboration on Jueneman's requirement for a "trusted sequence number." It is also more complicated and difficult to achieve than one might conclude from "sequence number."

4.12 NAMED TRANSACTION TYPE

The group noted that transaction type is the abstraction that is most similar to the concept of a TP in C-W. While this abstraction is named in some systems, it is often done in an operating system extension (such as IMS or CICS). The discussion group believes that this abstraction should be dealt with in a more general manner.

5. RECOMMENDATIONS

5.1 LINE BETWEEN APPLICATION AND SYSTEM

The group feels that work remains to be done on the requirements above to decide which must be in the system, which in the application, which can be in either, and what criteria apply.

5.2 ADDITIONAL ABSTRACTIONS

As noted under conclusions, the group feels that there is room for research in structuring the space between the name of the application and the name of the transaction type.

4.5 IMPLEMENTATION/MODELS - GROUP 5 REPORT

Prepared by Carl Landwehr

Carl Landwehr (NRL), Chair
Marshall Abrams (MITRE) Robert Baldwin (Tandem Comp.) Robert Courtney (RCI, Inc.) Betty Hill (World Bk) Robert Jueneman (CSC) Sushil Jajodia (George Mason U.) Stewart Kowalski (U. Stockholm, Sweden) Milan Kuchta (Defence Comm. Est., Canada) Stanley Kurzban (IBM) Donn Parker (SRI) Ravi Sandhu (Ohio State U.) Willis Ware (Rand)

1. CHARGE

III. Implementation/Models and the Integrity Framework Consensus

How does the (consensus) integrity framework relate to models of integrity policy and implementation approaches? Group 5 used this question to focus its discussions.

2. THE CHOSEN FRAMEWORK

The lack of structure in the consensus framework presented Thursday morning made it difficult to pursue this topic as intended, so the group agreed to explore modeling and implementation issues in relation to the definition of "property P" (the group 5 consensus on integrity). This definition can be stated as follows: X has property P if it meets specification Y with confidence Z. A possible instance of this definition is that data in an EDP system have integrity if they satisfy a specification of quality with adequate probability. Intrinsic data quality can be specified in terms of their timeliness, accuracy (correspondence with real world referents), and completeness. Extrinsic characteristics of data that contribute to its quality include their pedigree (explicit evidence of the origin of the data) and their robustness (resistance to corruption).

2.2 KURZBAN'S MODEL

Stan Kurzban first presented an access control model based on perceived commercial requirements. The model includes as elements user profiles, group (or "role-surrogate") profiles, resource profiles, programs (treated as a generalization of access modes), processes, and system-wide security data. This structure is specified informally and permits a flexible specification of user and group rights and access modes. Particular emphasis is placed on providing structures that model permissions related to organization responsibilities. This model, as well as those proposed earlier by group members Bob Jueneman and Ravi Sandhu, were included in the group's subsequent discussions.

2.3 WHAT IS AN INTEGRITY MODEL?

The group next turned its attention to the relation between integrity models and the attributes of data quality enumerated above. First, there was some discussion concerning the

meaning of "integrity policy" and "integrity model" -- the former is understood as shorthand for "a policy intended to establish, maintain, and/or enhance the quality of data in an EDP system," and the latter is understood as an abstract representation of a policy or system that is intended to display the aspects of system behavior that affect data quality and to suppress other aspects. The group listed as examples of models that might be used to model integrity Bell-LaPadula (B-L), Biba, Clark-Wilson (C-W), Harrison-Ruzzo-Ullman (HRU), Biba/Jueneman(B/J), Kurzban(SK), Sandhu (SPM), non-interference, and flow models. This list is intended to be representative, not complete.

The group discussed the structure, scope, and degree of formality of the models. All of the models considered in detail are oriented towards modeling and controlling the access subjects have to objects rather than the flows of data and information among objects. Considering whether particular models are appropriate for representing principles, policies, requirements, designs, or implementations, the group found that some of the models, notably Clark-Wilson, incorporate ideas from all of these domains, while Kurzban's model is primarily targeted toward design and implementation concepts. The formal access control models (Biba, HRU, SPM, Bell-LaPadula) seem to fit best into requirements or design contexts. In degree of formality, the HRU, B-L, Biba, and SPM are relatively formal, while the C-W and B/J are less formal. The SK model perhaps lies between these two groups in formality.

2.4 RELATING INTEGRITY MODELS TO THE FRAMEWORK

A key question to ask of an integrity model is whether it is complete with respect to an application: can it represent all of the quality attributes of concern in that application? Since there was no specific application under consideration, the group constructed a matrix relating all the quality attributes defined earlier to the models of interest. The results of this effort are shown in Figure 4.5-1. Since the matrix entries were derived through relatively brief group discussion, they are subject to revision, but the matrix structure should prove useful for comparing the properties of these and other potential integrity models.

As the matrix shows, all of the models can to some degree represent concerns about data accuracy. In most cases, however, the primary control on data accuracy comes from preventing unauthorized accesses to the data. In the Bell-LaPadula model, object contents are not represented, but maintaining the accuracy of the security label of an object is a primary concern. In the Biba/Jueneman model, timeliness can be addressed since timestamps are provided for, as are methods for associating an access history with a data item (thus providing a pedigree for it). This model also provides encrypted checksums for robustness. Clark-Wilson can model timeliness if a time period can be specified for the running of an Integrity Verification Procedure (IVP) against a Constrained Data Item (CDI). It addresses data accuracy and completeness through Transformation Programs (TPs) and IVPs.

2.5 RELATING IMPLEMENTATION MECHANISMS TO THE FRAMEWORK

The group next turned its attention to implementation considerations. The "common mechanisms" posed in the consensus framework were used as the basis for this discussion, and another table (Figure 4.5-2) was created to record the ability of each of these mechanisms to contribute to the maintenance of each of the attributes of data quality. One additional

mechanism, checksums, was added to the consensus list. Note that "consistency" has been omitted here as an attribute -- some of the group members felt that consistency was subsumed by accuracy and completeness, since it seems impossible for a set of data to be simultaneously accurate, complete, and inconsistent. As with the first matrix, no claim of completeness is made for this matrix but the structure may be of use to those wishing to study this problem further.

Evidently there are mechanisms listed that can address all of the specified quality attributes. This fact should not be taken as an indication that all the tools needed to solve the problem of assuring data integrity are in hand, however. Some of the mechanisms cited represent methods that are either abstractions of existing mechanisms (for example, Constrained Data Items and Trusted Transformation Processes may correlate with mechanisms existing in some database systems). Some, such as Separation of Duties, are principles that are widely recognized and applied manually by organizations, but lack support in most EDP systems. Although some confidentiality TCBs include support for the Biba integrity model, the Integrity Reference Monitor listed in the matrix is really a hypothetical entity. The application-specific reference monitor is in a similar state -- research on the NRL Secure Military Message System full-scale prototypes, for example, is striving to produce an application-specific reference monitor based on a security model that includes some integrity constraints on message fields, but such systems are not likely to be commercially available for some time. In fact, of the mechanisms listed, it is probably fair to say that only checksums, audit trails, and privilege mechanisms are routinely available for use in EDP systems today[3].

2.6 FURTHER CONSIDERATIONS

None of the group members believed that the matrices presented above were complete. If EDP systems with improved abilities to establish and maintain the integrity of the data they process are to be built, other considerations, including establishing ease of safe use as a key design objective and applying sound software engineering principles in their construction, must also be addressed.

3. RECOMMENDATIONS

To be able to build systems that better preserve the quality attributes of the data they process, the group recommends:

1. Further investigation of models that can represent a system's ability to preserve the property P,

2. Development of a glossary of integrity terminology to provide a common reference frame for work in this area, and

3. Study of mechanisms for flexible logging and processing of logs.

---

[3] The following is a counter comment by Donn Parker: "This ignores the many, commonly used controls for data integrity including sequence tagging, symbol and syntax tests, bounds controls, limit and range tests, separation, sample testing, checkpoint restarts, double entry, balancing, verification, consistency, and completeness tests."

4. QUESTIONS FOR FURTHER DISCUSSION

Stan Kurzban and Marshall Abrams have contributed the following list of specific questions that warrant further discussion/investigation in relation to integrity modeling:

1. What are the integrity-preserving principles developed by auditors?

2. Will generalizing "program" in the Clark/Wilson triple to include READ, WRITE, etc. work?

3. Must a model include a notion of a SAGA (a sequence of atomic transactions)?

4. Is a role merely a collection of privileges?

5. Are roles significantly different from groups, and if so, how?

6. Can a person execute only one role at a time? Does the integrity reference monitor need to know multiple roles which a person can assume? How is the change of roles implemented?

7. What are the consequences of allowing one individual to have multiple system identifiers?

8. What mechanisms are needed to reconcile conflicting permissions and denials?

9. How is the granting of privileges related to employee termination? Does the answer relate to whether granting of privilege is a role function?

10. Is there a restriction on membership in multiple groups?

11. Need a model describe hierarchies/networks of groups? Has a hierarchy of groups semantics (e.g., inheritance of permissions)?

12. Need categories be ORed, ANDed, ...?

13. How can the model include administration (e.g., take-grant, audit) permissions? Must it? Should it?

14. May one wear two "hats" in one session?

15. How should one model surrogates/agents/substitutes etc.?

16. Where should one begin (Biba, B-LP, Kurzban, C/W)?

17. Some authors have commented on the relationship of models, as in the ability to implement one model as a special case of another. Would it be worthwhile to investigate the relationship of models?

Quality Attribute \ Model	Bell LaPadula	Biba	Clark Wilson	Harrison Ruzzo Ullman	Biba Jueneman	Kurzban	Sandhu (SPM)
Timeliness			X		X		
Accuracy	* #	X	X	X	X	*	X
Completeness			X		X		
Consistency					X		
(Pedigree)					X		
(Robustness)					X		

X = yes
* = model can represent access to data by authorized individual
# = only accuracy of security label, not object contents, can be represented/controlled
Attributes labeled in parentheses are extrinsic, others intrinsic.

Figure 4.5-1. Can Quality Attribute X be represented/controlled in model Y?

\ Quality Mechanism\Attribute \	Timeliness	Accuracy	Complete-ness	(Pedigree)	(Robust-ness)
1. Attribute Measurement	X	X	X	X	X
2. Trusted Transformation Processes		X	?		
3. Constrained Data Items		X			
4. Object/Subject Authentication		X		X	
5. Event Signaling		X			
6. TCB (Confidentiality Only)		~ (labels)		X	?
7. TCB (Integrity Reference Monitor)		X	X	X	X
8. Application-Specific Reference Monitor	X	X	X	X	X
9. Separation of Duties		?	X	X	X
10. Privilege Mechanisms		X		X	X
11. Audit Trails, Logging				X	
12. Checksums		X			X

X = yes

Figure 2. Can the Specified Mechanism contribute to the maintenance of the corresponding Quality Attribute?

5. CONCLUSIONS, ISSUES, RECOMMENDATIONS

5.1 INTRODUCTION

This section presents a summary of the results of the Day-2 Working Groups. It is not intended as a substitute for the Day-2 reports themselves, but rather to provide the reader with an overview of the discussions.

The groups are presented in turn, beginning with the problem targeted by each group, along with a frame of reference (including definitions) where appropriate. The conclusions and recommendations of each group are then presented. In addition, the open issues identified by each group are listed. Where possible, the reason for deferral of discussion is noted (i.e, due to time constraints or disagreement). Finally, a summary of major areas of agreement and conflict are presented.

The five Day-2 Working Groups were assigned the areas of Operating Systems and Systems, Telecommunications, System Services, Applications, and Implementations/Models of Integrity Policy. These groups were led by Steve Lipner, Ted Lee, Grant Wagner, William Murray, and Carl Landwehr, respectively.

5.2 OPERATING SYSTEMS AND SYSTEMS (GROUP 1)

The Operating Systems and Systems Group began by selecting an integrity framework. They assumed the Clark/Wilson definition of integrity, defined mandatory as "no user choice", and defined discretionary as "user choice". Since a "secrecy TCB" might be a small subset of a system, but an "integrity TCB" might encompass the entire system, they chose to focus upon the security kernel that would control access, sequencing, etc.. They further decided to focus on integrity control objectives since these would then lead to the need for certain operating system features and in turn would lead to recommendations for guidelines and research topics. The selected operating system integrity features were divided into those that were necessary at the OS level and those that were optional (could be in the application or were not necessarily needed anywhere). The report then discussed several models that addressed some of the features and not others (Biba, Boebert, Sandhu) and continued on to characterize the necessary and optional OS features listed earlier.

The group recommended the creation of a set of integrity guidelines describing features of operating systems for meeting integrity control objectives. These guidelines were to also describe best practices and features in relation to environments. They also recommended against the creation of integrity criteria at this time. Such criteria should be delayed until after completionon of appropriate demonstration and reference implementation. Additionally, they recommended that the criteria separate functions and assurance (as in Sandhu).

Open issues requiring further research were identified as: separation of duties for privileged users; dynamic separation of duties; and enhanced user/manager interfaces to facilitate ease of correct use of control mechanisms.

They concluded that the operating system features they had identified mapped well onto the Clark/Wilson model but there were minor differences. For example, rather than supporting the Clark/Wilson user/program/data triple, a more granular approach was appropriate at the OS level, i.e., user/program and program/data binding.

5.3 TELECOMMUNICATIONS (GROUP 2)

The Telecommunications Group assumed a unique definition for data integrity. The primary goal of telecommunications is the communication of information between endpoints without modifying the information. Given this goal, the integrity policy for telecommunications can be stated as

"that received data comes from a known class of source, unchanged ... with an acceptable degree of confidence".

Some additional factors to consider are: 1) there exist applications which might accept changed data with an acceptable degree of confidence, 2) the use of received data may depend on what the receiver knows about the sender, and 3) an acceptable degree of confidence may vary with different components, e.g., class of source or data itself.

The Telecommunications Group compiled and described three lists addressing the technical problems in enforcement of this policy: 1) a list of threats and vulnerabilities of two types, accidental and malicious (Section 4); 2) a list of integrity mechanisms that respond to these threats (Section 5); and 3) a list of assurance measures for proper design and implementation of such mechanisms (Section 6).

They also compiled a list of non-technical barriers to widespread use of the known technology and a short list of open issues that were not addressed due to lack of time. Non-technical barriers included incompatibilityty of protocols and interfaces, lack of integrity adequacy guidance, lack of metrics for certification or evaluation of telecommunications integrity products, lack of policy on exportability of algorithms or equipment using them, conflicting rules about data flow across borders, and cost/performancece trade-off problems. Major open issues that were not addressed were identified as: oversimplification (thus leading to the need, for example, to assure the OS group covers distributed systems); interaction between systems and their interfaces so that integrity is preserved; the trade-offs for a high-integrity telecommunications system (e.g., between integrity and confidentiality); and the relevance of the Trusted Network Interpretation [TNI87].

The group concluded from their discussions that there was a need for guidance on Integrity Policy comparable to the TCSEC Environments Guideline (Yellow Book) [ENVIR85]. The group also concluded that adequate mechanisms exist to meet the technical requirements for enforcing the integrity policy for telecommunications. However, they perceive institutional and cultural barriers to wide-spread implementation and use.

5.4 SYSTEM SERVICES (GROUP 3)

The System Services Group felt their assigned area should include data base support, electronic mail, office automation, transaction processing, etc. but this was too broad for the time available. They therefore focused upon the relationship of data integrity issues to the underlying Data Base Management System controls. They examined how existing DBMS integrity controls implemented the Clark/Wilson notions of Independent Verification Procedure (IVP), Transformation Procedure (TP), Constrained Data Item (CDI), access triple, and separation of roles. The group agreed that DBMS query capabilities affected confidentiality while the update capability affected integrity. They thus focused on aspects of updates.

With regard to access triples the group agreed that most DBMSs provide two-component access controls (<user, data>, <user, transaction>, <transaction, data>) but could not agree on whether triples could be created from these. Precompiled transaction or canned query could possibly serve as triples in some DBMSs. The requirements for Transformation Procedures (TPs) include being able to bind names to TPs and being able to encapsulate CDIs so that they are only manipulated by appropriate TPs. Since it is then possible for two TPs to try to manipulate the same data, there may be a need for the triple to also include the context of the TP certification. Although it was agreed that the DBMS must support IVPs, no DBMS-based mechanisms for this were identified or proposed. All scenarios required reliable human reporting and intervention. The group agreed that separation of roles can be implemented with mechanisms commonly found in DBMSs, but that moderately complex constraints would be awkward and clumsy to maintain.

The group concluded that existing DBMS features could be used to implement separation of roles, but those features do not support easy maintenance and certification of those controls. The group also recommended that data definition and/or consistency check features could be enhanced to be more supportive of integrity controls and to allow separation of integrity controls from traditional data. Also, the group recommended that, early in the design of a DBMS, the proper balance between integrity and confidentiality concerns be arrived at, since later considerations of these issues might preclude being able to resolve them satisfactorily. Finally, the group recommendeded that more research and worked examples, where both integrity and confidentiality are needed, be performed to expand on available DBMS engineering compromises.

5.5 APPLICATIONS (GROUP 4)

The Applications Group was asked to examine the implications of data integrity requirements for applications. They restricted themselves to multi-user, multi-application systems. They also restricted themselves to the case where the boundary of trust encloses both the application and the data, but not the user. All other cases were excluded as having a trivial solution, or no problem. Finally, after examining the differences between tools and applications of those tools, the group concluded that the use of a tool was the same application as the tool itself, no matter the difference in the use (e.g., wp for inter-office memos vs. wp contracts).

The group reviewed principles and constructs. They stated that "the principle of least privilege is necessary to be able to make statements about the integrity of data." They also underscored the importance of separation of duties and segregation of functions. It was suggested that certain of the separations/segregations could be applied at the system level; others have heretofore been supported only by applications. They elaborated on the Clark/Wilson concept of the well-formed transaction.

The group concluded that it must be possible for management to control change consistent with the application requirements, that integrity policy was tightly coupled to the application itself, that data integrity cannot be guaranteed without system integrity, that an application should be accountable for the integrity of its data and that the integrity perimeter should be known to the application, that control of data should be localized both inside and outside the application, that there must be institutionalization of partitioning and contention in the system, and that there should not be any "all-privileged" users.

The group also concluded that application and transaction type are separated by too much space, and that additional layers of abstraction should be defined between them. They felt administrative and separation requirements could be better met by assigning them to the intervening layers. The group recommended additional work in this area.

The group elaborated upon Jueneman's list of requirements for data integrity. These requirements revolved around systems, applications and data issues and addressed immutability, attribution, logging and journaling, reporting and signalling, pedigree and provenance, validation, partition/compartmentation, identification, authorization, trusted date and time, synchronization and ordering, and names transaction type. The group recommended work be done on requirements to decide which must be in the system, which in the application, which can be in either, and what criteria to apply.

5.6 IMPLEMENTATIONS/MODELS (GROUP 5)

The Implementations/Models Group created yet another integrity definition. They defined a "property P" such that

"X has property P if it meets specification Y with confidence Z."

A possible instance of this definition is then that data have integrity if they satisfy a specification of quality with adequate probability. Intrinsic data quality can be specified via timeliness, accuracy, and completeness while extrinsic data quality relates to pedigree and robustness. The group used the model proposed by Kurzban as well as the Jueneman and Sandhu models for their discussions.

In order to answer the question "What is an integrity model?", the group defined an integrity policy as one intended to establish, maintain, and/or enhance the quality of data in an EDP system. An integrity model was then an abstract representation of a policy or system intended to display the aspects of system behavior that affect data quality and suppress other aspects. The group discussed the three models mentioned earlier and concluded they all

addressed controlling access to objects rather than the flows of data and information among objects.

They then examined seven models for how well the five quality attributes can be represented and controlled in each model and put their results in a matrix (Figure 4.5 - 1). Finally, they used a slightly modified version of the consensus mechanisms (see Section 3.2.2) to create a matrix specifying which of the five quality attributes are addressed by which mechanisms (Figure 4.5-2). The group noted that separation of duties lacks support in EDP systems today, that integrity reference monitors and application specific reference monitors are hypothetical entities, and that only checksums, audit trails, and privilege mechanisms are routinely available in EDP systems today.

The group stated three issues of concern: 1) the matrices are incomplete, 2) ease of safe use must be a key design objective, and 3) sound software engineering principles must be used in systemss that preserve integrity. The group also had three recommendations: 1) that there be further investigation of models representing the property P in a system; 2) develop a consistent glossary of integrity terminology; and 3) study mechanisms for flexible logging and processing of logs.

Two members of the group (Kurzban and Abrams) compiled a list of 17 open issues. These issues included such things as looking at the integrity preserving principles of auditors, whether a model should go down to the atomic transaction level, the proper characteristics of a role, and mechanisms for reconciling conflicting permissions and denials.

5.7 SUMMARY

Of the five groups, only the area of Telecommunication does not appear to require major technological advances. The field of telecommunications has been concerned with data integrity for a long time, and it shows. While they cautioned in their report that several issues needed to be addressed before "one could regard the subject as being close to complete", they also concluded that "protection of integrity of information...can be provided to essentially arbitrarily high levels of confidence without major technological advances." It is worth noting that only Telecommunications failed to identify items for further research. This is indicative of the maturity of the field of integrity in telecommunications.

Next closest to completion would be System Services' view of DBMS. They concluded that the existing DBMS features could be used for integrity control, but complex constraints would be difficult to implement, maintain and certify. Current features also do not allow effective separation of applications data and integrity control data. In short, technological advancement would be required for effective and flexible integrity control in a DBMS. (It is unclear whether this applies to system services in general, or only DBMS.)

Both Applications and Operating Systems appear to require technological advancement for integrity control. Existing features are insufficient or inappropriate. Both Applications and Operating Systems identified several areas where further research was required.

The Integrity Models Group appears furthest from meeting the goals. The theory and models of confidentiality simply do not apply well to the problem of integrity. In addition, this area has the most difficulty defining the question itself. It is difficult to evaluate models when there is no agreement on the definition for data integrity. The definition of "Property P" for the purpose of evaluating models appears to be a useful abstraction, though. Perhaps it may serve as an effective base for discussion until a single definition falls out.

5.8 MAJOR AREAS OF AGREEMENT

Three of the groups expounded upon the importance of separation of duties. Operating Systems identified static and dynamic separation of duties as research topics. The Applications Group suggested work was need to identify which separations or segregations could be enforced at the system level, and which could only be supported by the application(s). Systems Services based their report upon a model for enforcing separation of duties through a database management system.

Three groups expressed concern about interfaces. Operating Systems wanted them to facilitate correct use of control mechanisms, Implementations/Models wanted them to stress ease of safe use, and Telecommunications wanted them to preserve integrity.

Both Operating Systems and Telecommunications recommended the creation of a set of integrity guidelines. Operating Systems wanted the guidelines to describe features of operating systems for meeting integrity control objectives, to describe best practices, and to describe features in relation to environments. Telecommunications pointed to the TCSEC Environments Guidelines (Yellow Book) [ENVIR 85] as the appropriate model for their guidelines.

Two groups explored the possibility of implementing Clark/Wilson access triples as two pairs. System Services was unsure that this approach would work; Operating Systems proceeded upon the assumption that it would.

Several groups recommended the re-examination of the interaction between different parameters. Telecommunications identifiedd the cost/performance trade-off as a non-technical barrier, and identified the problem of (possibly) unresolvable trade-offs between design parameters (such as integrity, bandwidth, and economics) as an open issue. The System Services Group questioned the conventional wisdom (integrity and confidentiality are mutually exclusive), and recommended further research and worked examples so as to expand on currently available DBMS engineering compromises.

As stated earlier in this report (see Section 3.2), the Day-1 discussions on integrity framework elements did lead to some consensus on 1) policy and objectives for quality and 2) common mechanisms for quality. Although the five groups were unable to agree on a definition for data integrity, all felt that integrity was intimately related to quality.

5.9 MAJOR AREAS OF CONFLICT

Operating Systems was explicitly opposed to an Orange Book for Integrity. Although Telecommunications discussed mechanisms and assurances at length, the lack of metrics in this arena at this time also speaks against an Orange Book. The large number of areas targeted for research by the other groups would also indicate such criteria to be premature.

The data integrity framework discussions of Day-1 were unable to reach a consensus on the definition of data integrity. It appears that more work is needed in this area. See Section 2. for the discussion of this issue.

6. REFERENCES

BIBA77 Biba, K.J., Integrity Considerations for Secure Computer Systems, MITRE TR-3153, MITRE Corp., April 1977.

BOEB85 Boebert, W.E. and Kain, R., "A Practical Alternative to Hierarchical Integrity Policies," Proceedings of the 8th National Computer Security Conference, Sept. 1985.

CLARK87 Clark, David and Wilson, David, "A Comparison of Commercial and Military Computer Security Policies," Proceedings of 1987 IEEE Symposium on Security and Privacy.

CLARK89 Clark, David and Wilson, David, "Evolution of A Model for Computer Integrity", NIST Invitational Workshop on Data Integrity, January 1989. [In Appendix A]

ENVIR85 Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments, CSC-STD-002-85, National Computer Security Center, 1985. [Also known as the Yellow Book]

FIPS31 Guidelines for Physical Security and Risk Management, FIPS PUB 31, National Bureau of Standards, June 1974.

FIPS39 Glossary For Computer Systems Security, FIPS PUB 39, National Bureau of Standards, February 1974.

FIPS112 Password Usage, FIPS PUB 112, National Bureau of Standards, May 1985.

JEUN89 Jeuneman, R.R., "Integrity Controls for Military and Commercial Applications, II," NIST Invitational Workshop on Data Integrity, January 1989. [In Appendix A]

LIPNER82 Lipner, Steve, "Non-Discretionary Controls for Commercial Applications," Proceedings of 1982 IEEE Symposium on Security and Privacy.

LEE88 Lee, T.M.P., "Using Mandatory Integrity to Enforce 'Commercial' Security," Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 1988.

NIST160 Katzke, S.W. and Ruthberg, Z.G., Editors, Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), NIST Special Publication 500-160, January 1989.

PCIE88 Review of General Controls in Federal Computer Systems, President's Council on Integrity and Efficiency, October 1988.

SAND89 Sandhu, Ravi, "Terminology, Criteria, and System Architectures for Data Integrity," NIST Invitational Workshop on Data Integrity, January 1989. [In Appendix A]

SHOC88 Shockley, W.R., "Implementing Clark/Wilson Integrity Policy Using Current Technology," Proceedings of the 11th National Computer Security Conference, October 1988.

TCSEC Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, December 1985. [Also known as the Orange Book]

TNI87 Trusted Network Interpretation, NCSC-TG-005, Version 1, National Computer Security Center, July 1987. [Also known as the Red Book]

APPENDIX A

The Presented Papers

SOME INFORMAL COMMENTS ABOUT INTEGRITY AND THE INTEGRITY WORKSHOP

by

Robert H. Courtney, Jr.
Box 836
Port Ewen, New York 12466

It is the purpose of this paper to help set the stage for the Integrity Workshop to be held in January, 1989 and sponsored by the National Computer and Telecommunications Laboratory of the National Institute of Standards and Technology (formerly the Institute for Computer Sciences and Technology of the National Bureau of Standards). It is assumed that the readers are acquainted with the Clark and Wilson paper(1) and are aware of the initial integrity workshop held at Bentley College in October, 1987(2).

The high level of interest provoked by the Clark and Wilson paper led to the convening of that first Integrity Workshop. Many data processing professionals see in that paper the genesis of a rational, systematic and comprehensive approach to needed improvement in data quality. There is also rather broad agreement that the paper was particularly timely because it appeared when interest in more systematic approaches to the accuracy, completeness and timeliness of data was increasing fairly rapidly and becoming more broadly based than it had been in the past.

There were two things about the Clark and Wilson paper which caught my attention immediately. First, I had no firm notion of what the authors meant by the word "integrity". The paper didn't define the term, but there was a clear implication that accuracy and completeness of data were the primary goals.

Second, the paper posed something of a challenge. It provided an orderly explication of some sound principles of potential value to those seeking improved data quality and assurance of that quality. But the set of principles provided there, while quite valuable, was not complete. This is not a criticism of the paper. It was clearly not the authors' intent to define a set of conditions which were both necessary and sufficient for the attainment of uniformly adequate data integrity. Nevertheless, there seemed a clear need for the identification and dissemination of the other principles not contained in the paper.

THE INTEGRITY WORKING GROUP.

In February, 1988, an Integrity Working Group, chaired by the author, was established by the National Institute of Standards and Technology's Computer and Telecommunications Security Council(3). The principal purpose of the group was to lay a solid foundation for the upcoming Integrity Workshop. The initial tasks selected by that group were the following:

1. The establishment of a suitable definition for the word integrity and

2. Setting bounds on the scope of the topics to be addressed by the Integrity Workshop. That was dependent upon the development of an orderly and systematic perspective of data integrity which would hold reasonable promise of providing both a foundation for future work and at the same time offer hope of the earliest practicable applicability of the work being done.

This paper, then, provides an informal description of the early activities of that Integrity Working Group. The Working Group hopes to encourage others to provide thoughtful critiques of what they have accomplished and to invite suggestions of direction and redirection.

INTEGRITY, A PROPOSED DEFINITION.

The definition of "integrity" agreed upon by the Integrity Working Group is as follows(4):

> Integrity -- The property that data, an information process, computer equipment, and/or software, people, etc., or any collection of these entities, meet an *a priori* expectation of quality that is satisfactory and adequate in some specific circumstance. The attributes of quality can be general in nature and implied by the context of a discussion; or specific and in terms of some intended usage or application.

For several years now, Dr. Willis Ware, of the RAND Corporation, and I have been urging more care and attention to the terminology used by data processing people(5). Data processing is a field in which both precision of meaning and ease of communication are important. For that reason, the distortion of the meanings of good, useful, and commonly used words is to be abhorred as is the invention of any unneeded and, consequently, poorly understood new words.

The data processing vocabulary has seen a much needed stabilization over the past few years, but the terms used and misused by those involved in computer security area for the most part have not been beneficiaries of this. Confusion in terminology is now interfering with the evolution of suitably orderly and systematic approaches to computer security.

Both Dr. Ware and I, quite independently, had focussed on the word "integrity" as one particularly deserving of better understanding and more uniform definition. Thus, we both welcomed an opportunity to join with others in the working group to propose a definition of integrity suitable to the data processing environment but which is not in conflict with the definitions found in the more widely used dictionaries.

For the past seventeen years, I have found it desirable and often essential that I explain to people with whom I am working or to whom I am talking exactly what I meant by a number of security-related terms. Integrity has always been among them.

I have tried to convey two notions of integrity. First, the integrity of an object is not dependent upon its relative goodness, that is, its quality. Second, the integrity of an object is dependent only on its freedom from unpleasant surprise; that its quality is no worse than it is thought to be. This is to say that the quality does not have to be high for an object to have integrity. It is required only that the quality of the object be known and that, whatever

its limitations, it is useful for its intended purpose. It is not to say that the quality is as good as we might like or that it is not desirable that it be better. To have integrity it need only be no worse than we thought it was. This is to say, in Willis Ware's terms, that the quality of the data meets a priori expectations.

The agreement of the Integrity Working Group on a definition for integrity was reached only after many exchanges of proposals, their revision, and the consideration of comments offered by people from outside that group. Notable among that latter group was Donn B. Parker, of SRI. His patience with our efforts is greatly appreciated.

The task of arriving at a suitable definition for integrity was difficult because the word has been used to mean so many different things that it became necessary to consider the availability of other, commonly used words with generally accepted meanings to fill the gaps that would be left when we proposed a more constrained definition for integrity.

Many people have used integrity to describe the accuracy, timeliness, completeness and other such attributes of data. We do not accept that use of the word. We specifically exclude the use of quality and integrity as synonyms. The word "quality" should be used to describe the goodness" of the attributes of an object which are meaningful to some particular application of it.

It is highly desirable that the meaning of integrity applied to any object also apply to all objects and, particularly all objects in the data processing field, including data, systems, programs, devices, and people, and collections of these.

Most of us have been forced to work with data which are not as good as we want them to be but which are at least adequate to our needs. Such data have integrity if we understand the limitations on their quality. Similarly, we have all learned that we can do at least acceptable work with less-than-great people provided only that we know what their limitations are. If we know the limitations on the quality of our data and of our people and of other things on which we are dependent, we can usually operate quite well.

If we do not know the limitations on the quality of those things on which we are dependent and, as a consequence, we place trust in them in excess of that deserved by their quality, we may be assuming far greater risk than we anticipated or would find acceptable. Thus, the data and the people and the other needed objects have integrity when their quality, whatever it might be, meets our expectations even though it might fall far short of our desires.

As we define integrity here, it has only two states. There are no degrees of integrity. We either have it or we don't have it.

Integrity reflects trust in quality. There can, however, be degrees of trust.

"Quality" encompasses all of the attributes pertinent to an assessment of the goodness of an object relative to a particular application environment. Quality can vary. It can degrade without loss of integrity provided only that there is no expectation that the quality is greater than it is.

Quality(6) is usually best described in terms of the relative goodness of each of its component attributes. For example, if the usefulness of a body of data in a specific application is dependent upon the adequacy of its accuracy, timeliness, and completeness, the quality of the data as a whole cannot be assessed except in terms of the quality of each those operable attributes.

Quality is not synonymous with utility or value. Under some circumstances, quality can be changed without corresponding changes in utility or value.

Finally, to repeat for emphasis, integrity relates to our expectations of quality. The word should not be used to express the quality, value, utility, or security of an object.

BOUNDING THE WORKSHOP SCOPE.

Consider the following assertions.

1. Information and data are not synonymous. Data are a collection of facts. Information is derived from data in response to a particular need.

2. Data have value (worth) which is measurable only in terms of value of the uses which can be made of those data. Data have no intrinsic value(7). The value of data has no direct relationship to their initial cost or their replacement cost(8).

3. The particular attributes of data which determine their value are a function of the uses made of those data. Accuracy, completeness, and timeliness are commonly encountered attributes, but there are often others equally important to specific applications(9).

4. The value of data does not necessarily increase with each incremental increase in their quality. There are often instances where the cost of improved data quality exceeds the value of that further enhancement. Thus, limitless improvements in data quality are not always justified.

5. The quality of data needed to satisfy specific applications should be evaluated, attribute by attribute, to establish the cost-benefit relationships between quality and value. If the threshold is set higher than is actually needed, the cost of data to an application may seem too great to justify such use when, in fact, less costly quality would be adequate.

6. The attainment of requisite data integrity and data quality should be guided by specific statements of policy.

7. A policy should be no more than a statement of the principles which will guide the conduct of functions governed by that policy. A policy should not include procedures and practices.

8. There is a large body of principles from among which those pertinent to any particular application environment can be selected for incorporation into a specific policy statement.

9. There is a need to identify as many as possible of those principles as might be of sufficiently general benefit to warrant their inclusion in a list of such principles from which formulators of policy can select, cafeteria-style, those appropriate to their needs.

10. There is a need to develop better understandings of the appropriate roles, relative to data quality and data integrity, of Quality Control, Quality Assurance, Data Base Administration, the MIS management, the functional area managers supported by the data, the internal audit functions, the external auditors, and the computer security group.

We suggest that authors submitting papers for the Integrity Workshop select their topics from the ten issues listed immediately above. We do not believe that authors will be adversely constrained by these bounds. We hope that our provision of such gross bounds will increase the probability that the papers submitted will be mutually supportive and offer promise of earlier availability of useful products in an area where such products are sorely needed.

A Note from the Author:

I sincerely hope that the work of the Integrity Working Group during its relatively brief life will provoke a blizzard of constructive controversy leading to a far more broadly based understanding of the issues involved. It is also hoped that this will lead to a wholly rational and systematic approach to data integrity and quality, and to a better understanding of the appropriate roles for the many organizational entities which have interest or involvement in these matters.

Bob

APPENDIX I

The following is a message from Dr. Willis Ware, RAND Corporation, to the other members of the Integrity Working Group. It is included here because it provides the definition of integrity on which that group has agreed and also includes some explanatory material which should be of interest.

****

Subject: DEFINITION of INTEGRITY
Date: Thu, 18 Aug 88 14:15:12 PDT

To: All

After many agonizing messages back and forth and sometimes including others, Bob Courtney and I have zeroed in on and agree on the following definition of integrity. We have had several variations of this, and it now represents some fundamental decisions and reflects some already debated points.

So before, you jump us read the supporting comments below.

Integrity -- The property that data, an information process, computer equipment and/or software, people, etc. or any collection of these entities meet an a priori expectation of quality that is satisfactory and adequate in some circumstance. The attributes of quality can be general in nature and implied by the context of the discussion; or specific and in terms of some intended usage or application.

SUPPORTING COMMENTS:

1. We did intentionally define the term broadly, rather than restricting it to data integrity.

2. We did intentionally retain the concept of quality as a collective term for the set of attributes against which integrity is to be judged. It is obviously possible to swallow the concept of quality into the definition of integrity, but we thought it useful to define a breakpoint between the two concepts and to have a special term for "set of attributes."

3. We did intentionally retain the adjective a priori even though it is slightly redundant in conjunction with expectation. We thought it added extra emphasis that the question "does it have integrity?" is an up-front before-the-fact one to be asked.

4. We did intentionally frame it to be useable in a very general context [e.g., talking with managers] or in a very specific context [e.g., data to be used for some application]. We rejected the phrase "... for some intended purpose" as being too restrictive.

5. We do believe that integrity, as a property, is binary; you "got it" or "you don't."

In this connection recall the item I circulated in April to this group. It was the piece I had submitted to Ted Lee's Security Forum in October 1984. Recall that therein I distinguished very carefully between integrity as the property, and our ability to make a judgement about the measure of the quality. In brief, one may decide that something does/doesn't have integrity, but his confidence in the decision may not be absolute. Thus, the confidence level in the decision about determination of presence/absence of integrity is indeed continuous in nature and might well be called the "level of trust" in the decision that has been made.

6. Finally, we believe that this definition is both consistent and congruent with the ideas and concepts proposed in Clark- Wilson.

As Bob said in the penultimate message about this: "Let's move on to the next problem."

==========

Just to remind you, I am wrapping up the report of a NRC/DOE committee that I chaired, and it will contain footnotes that are consistent with the definition above, and will carefully point out the difference between integrity as a property, and integrity policy (which would better be called "policy on the assurance of integrity").

I had sent early version of these footnotes to you, and will forward the revisions to anyone that's interested. Suffice it to say that the definition above and the DOE report will track precisely.

willis ware

APPENDIX II

The following is material written in October, 1984 by Dr. Willis Ware, of the RAND Corporation,, for Ted Lee's Computer Security Forum. It provides useful background material for readers who want to be fully aware of key prior activities in the data integrity/data quality areas.

Date: 1 October 1984 13:26 edt
From: Willis Ware <willis at RAND-UNIX>
Subject: [Integrity and Trust]

INTRODUCTION

Below I want to offer some views on the dialogue that has appeared in recent issues of the Forum with regard to "integrity" and its measurement. It may be too late to persuade anyone that the words should be used differently, but at least I'll put my views on record. First I think it important to indicate where I'm coming from.

I feel that the computing field has done itself a disservice from to time over the years by being careless about its specialized terminology, being inconsistent in terminology, and even in inventing terms when perfectly adequate other ones existed. It was clearly much worse in this respect 25 years ago; today standard definitions have appeared, professional societies have fostered standard usage through their publication and editing policies, etc.

When the Defense Science Board project was done at the turn of the 70s,*
- ----
*[available now as an unclassified publication: Security Controls for Computer Systems, Report of the Defense Science Board Task Force on Computer Security; published as a classified document, January 1970 and republished as an unclassified document, October 1979.]
- ----
we paid very careful attention to defining our terms, to using ones that were known to traditional security people, etc. The "Orange Book"*
- ----
*[DoD Trusted Computer System Evaluation Criteria, 15 Aug 83, Computer Security Center, National Security Agency.]
- ----
has seemingly introduced new words for terms that had already been well defined, and that are at variance with what were introduced in the DSB report and had been a part of policy dialogue in government for most of a decade. It seemed so unnecessary.

==========

BACKGROUND

The computer-security field is really emergent in the sense that it is moving from the R&D closets of the 70s into the harsh world of reality where it will have to compete for funding and status with a lot of other things. It has no automatic call to acceptance; it will have to make its way with clarity of expression to those it will seek to influence. Thus, I would hate to see its acceptance deterred because of invention of unnecessary terms, of careless and inconsistent use of terms, of confusion in its communication.

As a historical observation, the research community that has worked the computer security topic over the years -- especially its software aspects - -- developed a very specialized vocabulary within itself, and its terminology has now emerged into the world at large because of the existence of the Computer Security Center and its Orange Book of Criteria. But the world has trouble understanding very specialized words --especially when they have widely accepted and established extant meaning. I suspect that most of the world will have trouble reading the Orange Book intelligently. Regrettably it does not contain a set of standard terms with well thought-out definitions. The glossary in the back is not adequate for most readers.

Ah, though, the world must understand the CSC, and the Orange Book, and the way the CSC will talk and influence the world. Especially, managers/ administrators/operators of computer centers/programmatic people/vendors/ system designers/system specifiers/contract negotiators/ etc. have to realize what computer security is all about and they all have to be able to talk among themselves consistently and clearly. They must understand it or the whole CSC thrust will have trouble gaining acceptance and support.

While one can comment critically about the contents of the Orange Book, its substance is good and it makes a very essential contribution to getting on with system security and eventually network security. It is a first edition and of course, can be expected to have shortfalls (as suggested above) that will be taken care of later.

==========

THE ISSUE

I hope that my comments and position do not cause misunderstanding about where I stand. The way I perceive the Forum discussion is that people are casting about looking for ways to rate the processes which are components of a trusted system, whether those components are software processes (other than ones embedded in the operating system), or (at least in principle) are people functioning within the system as operational individuals or users.

I am not in the least contesting the need to attribute a quality measure to processes in or components of a trusted system which we expect to enforce a security policy. There is no question but that such a rating system could be useful and essential.

What I am contesting (below) is the particular word that has sprung up to describe the desired quality and how to measure it. If I hear the discussion right, at least some group of people want to call the quality "integrity" and they want to measure it as "levels of integrity". And

I think that's wrong. And some of the terms proposed to measure it are even more troublesome.

In any event, when I bumped into the discussion of "integrity levels" in the Forum, I said to myself "They're at it again, using words in strange ways that non-specialists will not understand. And it need not be." I don't know what phrase should be used to describe measures of quality but I have a strong conviction that it must NOT be "levels of integrity" for the reasons set forth below.

We need eventually to be able to measure trustworthiness -- hopefully in a quantitative way -- at many places in a system which we call "trusted". There are bound to be instances in which we'll want to say something more precise than simply a "system is trusted" at the global level.

As a collateral observation, I'll assert that we would do well to clearly state as an up-front requirements statement exactly what we expect a system to be trusted for, so to speak its functional aspects of trust. Example: in a communications controller, it must be trusted (among other things) not to corrupt the security parameters and not to perturb the association between a message and its security parameters.

==========

A DIFFERENT APPROACH

I seem to have tuned in late on the current discussion of terminology; or perhaps I had simply not noticed it going on. Since computer security is an issue of high interest to me, I'll chime in.

My thesis is straightforward.

The Center in its Orange Book has established a sequence of Criteria against which to judge quality of computer products in terms of their ability to enforce security and access controls. [See note below for two that have been rated.] Such rated products will become part of a system which will be called trusted. What can be more reasonable, therefore, than to speak of "level of trust" for the components of a trusted system -- just as one will speak of the level of trust of the overall system?

In terms of the Center's procedures, system components will be rated on a scale from D to A-1. [See below for further discussion of the point.] The overall judgement about trustedness of a system will be in terms of "accreditation" which measure will be in terms of the sensitivity of the information that the system is allowed to handle. While "accreditation" is not precisely a measure of trustedness, it nonetheless reflects the level of trust that emerges from a prescribed procedure; at minimum accreditation will become a surrogate measure of trustedness until something better comes along.

The Center's established approach uses one scale (D through A-1) for rating a component, but a different scale (sensitivity of information contained) for rating the system.

TERMINOLOGY

There are terms (some were suggested in the Forum) that must not be used to rate software processes. The terms CONFIDENTIAL, SECRET, and TOP SECRET all have definitions in an Executive Order in terms of the damage to the country that would be anticipated if information of a given classification were revealed to an unfriendly opponent. They are hierarchical with TOP SECRET being the most restricted category and also, that with the gravest potential damage to the country.

The same 3 labels are also used to denote CLEARANCES of people; they are really defined implicitly and by reflection; e.g., an individual cleared to SECRET is authorized in principle to have access to information which is classified SECRET or lower. If his job responsibilities requires that he needs access to some specific body of SECRET information, then, by administrative action, he is given NEED TO KNOW for it. There are investigative procedures that have been established for granting of various kinds of clearances; they are presumed to establish the degree of trustworthiness of an individual and to do so with the required degree of confidence, or certainty. A clearance really measures the level of trust for a particular person that some administrative authority is willing to accept and certify. For example, one might be trusted to deal with SECRET information, but not TOP SECRET.

So, whatever we (as a computer fraternity) do, the meanings of the these 3 words in both usages are fixed and cannot be tampered with, and probably should not be extrapolated to other usages that will only cause confusion.

Next, INTEGRITY is a hard word to define. Various attempts have been made over the years. One was: The property of being free from surprise. Another: [Something has integrity if] it is what it is supposed to be. Admittedly these were created at a time when the word was applied primarily to data or databases or files. The idea was that such things would be what they were expected to be, had not been trifled with, had not been contaminated accidentally, etc. This usage is quite consistent with the dictionary definitions which generally talk about "an unimpaired condition, soundness, state of being complete or undivided, uncorruptability" and giving "duplicity" as an antonym. Interestingly, the Orange Book from the Computer Security Center does not define the term.

Usage of the term is largely binary in common parlance. Something has integrity or it does not; we don't speak of "65% integrity". While I would be the first to agree that language lives and changes, let's not carelessly or thoughtlessly create a subtle new and different usage for a term that is widely understood and used. Especially let's not create a new usage that is largely diametrically opposed to established use. This is particularly important for that large group of managers, administrators, ordinary computer folks, etc. that do NOT share the inner workings, thoughts, concepts, constructs, etc. of the quite small technical computer security community; but must know what it is all about.

Next, TRUST. There is not a really adequate definition of it, although the Orange Book has one. It speaks of "employing sufficient hardware and software integrity measures .... for processing simultaneously..." [This was originated by Steve Walker when he was in the OSD,

and he is aware of my concern about it.] Only multi-headed machines or some of the big vector computers process simultaneously; therefore, the right word is really "concurrently".

However, what is an integrity measure? Whatever it is, it has integrity or it doesn't -- given the binary nature of the term as commonly used. If it has it, then it is free from surprise and it is what it is expected to be. A better word is certainly "safeguards" or "security safeguards". With this understanding, then the Orange Book definition simply says that the system is supposed to have the right set of safeguards, and that they are to be what they are supposed to be; i.e., correct manifestations (or implementations) of some functionally defined entity.

The major flaw in the definition is that it fails to say anything to the effect that the system with its surprise-free safeguards is supposed to do its thing with a certain level of confidence, or with a certain probability of success, or with a certain warranty, or with a certain guarantee, or to oppose a defined threat, ...

What is missing is just what the Forum discussion is evidently searching for. How trusted is a trusted system?

It is "LEVEL OF TRUST" or "TRUSTEDNESS" that everyone wants to measure, not level of integrity. First of all, trust is not a binary attribute as the word is commonly used. It is understood that someone (or some thing) is trusted for some purpose (but not others); we are all accustomed to estimating degree of trust in our daily lives. We often say: "Can I trust someone (or some thing) for ...."; thus, we make a value judgment in terms of some end goal, and implicitly the goal has a threat attached to it. This is precisely what trusted systems are all about; we make a judgment (the CSC calls its certification) that some hardware/software implementation -- in the long run the operational environment for it will also have to be specified -- will impose appropriate security rules (security policy or access control rules in the language of the computer security community) in spite of being faced with some threat.

It is in the characterization of threat that the defense and commercial communities differ significantly. The defense community has to assume that its threat is a well financed, intellectually well based, resource rich, etc. one mounted by a formidable military (or foreign policy) opponent. After all, the world has centuries of experience with intelligence affairs, counter- intelligence protections, behavior of sovereign powers toward one another... on which to base such an assumption about threat.

The commercial world can make its own characterization of threat, and that's what risk analysis is all about -- as discussed in the NBS FIPS on the subject.

FINAL POSITION

So here's my bottom line after all the discussion.

o Do not use the phrase "integrity levels" as the metric for processes in or components of a trusted system. To introduce such a concept would add confusion to the already difficult and complex issue of computer security, and after that, network security.

o Agree that TRUSTEDNESS is the property that has to be quantized, or estimated, or measured on some scale, throughout a system from people to component to overall system.

o Acknowledge that TRUSTEDNESS will have to be measured against some scale at many levels throughout a system; the scale may be but need not be the same. The "scale" may not be a numerical one but rather one such as implied by the Criteria and the various levels of certification as specified in the Orange Book. For people it already is embodied in the process of granting clearances.

o Do NOT adopt the terms CONFIDENTIAL, SECRET, TOP SECRET for levels of trust. They are already spoken for, and have widely used accepted definitions.

o Do NOT employ the notion of measuring trust for people when the context is the defense world. The trustedness of people is already implicitly measured in the granting of a clearance. A SECRET cleared person is trusted to have access to SECRET classified material, and thereby not incur the corresponding damage for the country.

o TRUSTEDNESS OF PEOPLE is a legitimate concept when the context is the non-defense, non-federal-government commercial/industrial/academic world. It is already in use in some places; some organizations have defined the level of trustworthiness required of individuals who fill jobs considered to be sensitive; and therefore, potentially dangerous to the interests of the organization if the individual-in-the-job divulges information. Interestingly the "level of trust" is normally defined implicitly in terms of the background investigation that an individual is required to pass.

It's not hard to figure out that I disagree with the proposals outlined in the Forum recently, and especially with some of the terms suggested for measuring quality.

Given my convictions as expressed above, clearly one should not measure TRUSTEDNESS with a term which uses the same root -- "trustworthy" and "trusted" are no-no's. Don't for a moment commingle the notions of "clearance" and "integrity"; they have nothing to do with one another, and to mix them up will only create major confusion. Any other terms might do - -- those suggested by Dorothy Denning, the other ones suggested by Kahn and Millen. Whatever ones are chosen, make sure that they fit into accepted usage of the words in other contexts, and that they clearly do not conflict with established usage, and especially, that they are not ambiguous as to the defense vs. the commercial
world.

==========

A RELATED ASPECT

There is a very different but very important aspect of this whole dialogue. Here it is.

The Orange Book sets forth a series of evaluation Criteria from D through C-1 through A-1. D is a commercial run-of-the-mill product whereas A-1 is a software product built to carefully

controlled specifications by a carefully controlled process by trusted people in a carefully controlled environment. This might well be translated in the military environment into cleared people in a classified environment; for the commercial system, into people whose trust have been established somehow working in a restricted and controlled environment. While the text never says it quite explicitly, the criteria are really an ascending scale of trustedness. A C-1 software product has a minimum level of trust; an A-1 is most trusted. Some day the scale may go beyond A-1. On some numeric scale starting at zero, a D software product would have a trust value of 0.

For the reader of the Forum who is not familiar with the Criteria, the technical substance of them is a result of a long dialogue among informed technically competent people who understand the defense environment, who in the large have been computer security practitioners for many years (especially researchers), who had a feel for the operational environments of such systems, and who understood what was reasonable to ask for in terms of vendors being able to respond. The criteria are not wild dreams of entrenched interests; they reflect the considered technical judgment of experienced people.

In any discussion of software security safeguards, it is important to note that the "Criteria" is the only game in town; it is the mechanism by which the Federal government will measure the products that it buys and influence vendors to produce commercial products that measure up. Thus, in the long run, the Criteria will set the tone for software safeguards, for the creation of trusted software, and for evaluating/certifying it. And the Criteria will de facto set the tone for discussion of software security in commercial/industrial circles. Thus, it makes no sense to go off in other directions with words, measures of quality, etc.; it will be fruitless because the Criteria have preempted the ball game.

[As a matter of general interest, two products have been rated: IBM's RACF version 1 release 5 is C-1; ACF2 release 3.1.3 is C-2 (which is higher than C-1). Evaluation summaries of both are available from the Center.]

Therefore, coming back to the main point, if the Criteria were widely known and if all software products about which we might wish to discourse had been evaluated by the CSC, a perfectly acceptable and desirable scale would be D through A-1. The two just mentioned were rated on just that scale!

INDEPENDENT EVALUATION

Unfortunately, not all products will have been evaluated by the Center for quite sometime to come, so we might need some other subjective but hierarchical scale against which to rate things; that's what the Forum discussion is after. But the big questions will become: Who makes the judgement about level of trustedness -- whatever the scale is? And how is the judgment reached? And against what criteria?

If it's done in less than a carefully controlled and defined process by competent people having access to extensive facilities equipped with a variety of test tools, the ratings will be

meaningless. We will have another tower of Babel but this time with respect to trustedness, rather than programming languages.

A PROPOSAL

Thus, I might take the view that the dialogue to which I have contributed is a vacuous one that will have little import. Unless, that is, somebody gets busy (like the Institute of Computer Science and Technology at the National Bureau of Standards) and establishes procedures that are analogous to the Center's, are correlated with and carefully tied to them, and can be carried out by suitably equipped people (intellectually and skillwise) in an appropriate environment.

It may well be that we need an NBS Federal Information Processing Standard for the evaluation and certification of the "level of trust" of a trusted system. There is precedent for a co-operative effort between the NBS and NSA -- it's the Data Encryption Standard.

The ICST/NBS has in effect done a similar thing for risk analysis; there is a document that tells anyone that wishes to do so how to carry out a risk analysis. A companion document that did a similar thing for rating a software package against the Center's criteria would be very useful to bring order out of potential chaos -- even though the resultant rating would have to be carefully phrased in some such fashion as (say) "equivalent to a B-1 CSC evaluation" -- or some other phrase to that effect. "FIPS Criteria" need not be exactly the same as the Center's Criteria, but the two would have to be consistent although the former could be adapted to the needs of unclassified (e.g., civil government, industry) users.

One would have to be careful with the words to avoid implying that the Center had actually rated some item that had been measured against a FIPS Criteria. At the same time the words should convey the equivalent Center-rating that the doer-of-the-rating believes valid -- assuming that he had used some standardized process established by the ICST/NBS.

My suggestion might be seen as dangerous by the Center because it can be seen as an intrusion on its charter. At the same time, though, we -- the country -- cannot afford to have everyone running around putting trustedness evaluations on software by means that are not-standardized, not described, and therefore not useful but, on the contrary, confusing and misleading.

It's a new idea and approach -- whatever other problems it might raise.

Willis H. Ware
Rand Corporation
Santa Monica, CA.

- ----------
End of COMPUTER SECURITY FORUM ARTICLE
****

The following was appended to the article when a copy was sent by E-Mail to the author in April, 1988 in support of Dr. Ware's position at one juncture in our many and sometimes rather heated E-Mail exchanges.

****

OK, so you know where I'm coming from and what I think about words. Let me recast the whole thing in different terms and then try on some examples which have occurred in the dialogue already.

My argument goes like this:

1. We have an a priori expectation that something has some property (or set of properties). E.g., a hull is without holes; an airframe is without cracks; data is without error.

2. If we investigate the expectation that we bring to the situation, we find that it is true or it is not. If true, we say that situation has integrity; if not true, it does not have integrity. E.g., an airframe without cracks or breaks has structural integrity after we have investigated the situation; the data does indeed have the qualities of accuracy and completeness that we a priori expected.

3. Integrity is a binary quality, and common usage is just that way.

4. But suppose that we do not have opportunity to make a full investigation, perhaps only a partial one. We still must make a judgment of integrity -- yes or no -- but on incomplete evidence. We do indeed make the decision and we decide something has integrity or not, but admittedly there is uncertainty. BUT the uncertainty is with the decision which we made, not with the property about which we made the decision.

5. We may well have decided incorrectly, but the question we ask ourselves is: "Do we trust our judgment?" If we do (we feel that it is a process with high assurance measures and will function as we believe it should), then we attach a high confidence to our decision about integrity of the something. Whatever the case, the potential uncertainty is with the estimate of whether integrity is 0 or 1, not whether integrity has some other fractional value. The uncertainty is never with the property which we call "integrity" per se. Quite the contrary the uncertainty equates to the degree of trust that we attach to our decision-making process.

6. "Trust" then is the quality that we somehow must measure in various circumstances of a secure computing system. It will have to be applied to processes, to paths, to people, to components ..... and to the overall system. The notion of "trust" as promulgated by the Center is very narrow and inadequate for delineating and describing the full scope of secure computer systems.

Final example. We decide that data has integrity; it has all the qualities that our a priori expectation led us to believe. A trusted process then operates on that data; whatever the process does, we believe that it will do so as defined and with high confidence. The process

may well upset some of the qualities on which we based our integrity decision; e.g., the process may intentionally make the data incomplete or inaccurate.

Does the data have integrity after the process has completed? We know what the process is supposed to do to the data. If indeed the data has the a priori (i.e., as measured before the process functioned) expectation for the data exiting the process, then it has integrity; otherwise not. Thus, the misbehavior of a trusted process can in fact destroy integrity.

In terms of the discussion that I had 4 years ago with the Orange Believers, they didn't understand the details of the issue they were discussing. They were not in search of "integrity measures"; they were in search of "measures for the process that determines whether or not something has integrity" -- loosely speaking, measures of integrity not integrity measures.

OK -- where are the flaws in the argument? The loopholes? The inconsistencies?

ENDNOTES

1. A Comparison of Commercial and Military Computer Security Policies. David D. Clark and David R. Wilson. IEEE Computer Security Conference, 1987.

2. Report of the Invitational Workshop on Integrity Policy in Computer Information Systems, Bentley College, October 27-29, 1987, by National Bureau of Standards Institute for Computer Sciences and Technology, 1988.

3. The members of the Integrity Working Group are, in addition to the author, Dr. Willis Ware, RAND Corporation; Mr. Steven Lipner, DEC; Mr. Stanley Kurzban, IBM; Mr. Peter S. Browne, Profile Analysis Corporation; and Mr. Carl E. Landwehr, Naval Research Laboratory. Dr. Sylvan Pinsky joined the group in September, 1988.

4. The text of Dr. Ware's message to the Integrity Working Group and a few others conveying this definition and some comments on it are included here as Appendix I.

5. A copy of Dr. Ware's paper, Integrity and Trust, which appeared in Ted Lee's Computer Security Forum of October 8, 1984 is included here as Appendix II.

6. The next task for the Integrity Working Group is to address the definition of quality as it relates toobjcets in the data processing environment and, more specifically, to data security.

7. It can be argued that value cannot be an intrinsic attribute of anything because value is a relative thing and it must be relative to something external to the object itself.

8. The commonly encountered notion that data have intrinsic value or that the value of data bears some relationship to the initial or replacement cost presents a significant barrier to many data security folks attempting even semi-quantitative risk analyses. We get about fifty to sixty calls per year from people who have encountered that problem in their attempts to cost-justify security measures. Most often, their problems are solved when they are persuaded that they

should consider the value of the functions supported by those data or, conversely, the cost of not having the support of those data.

9. It is regrettable that the achievement of data integrity and data quality has been so separated from the provision of data confidentiality in the minds of many people working in the computer security area that the suggestion that the confidentiality of data, or the lack of it, greatly effects the quality and value of those data as much as do other, more commonly accepted attributes of quality. Too often the result is failure to recognize the applicability of many controls to both confidentiality problems and to integrity and quality concerns. The notion of that separation seems so ingrained now that we have deemed it unwise to combine any effort in changing that notion with the other changes we have proposed here.

THE END

Evolution of A Model for Computer Integrity

by

David D. Clark
Senior Research Scientist
MIT Laboratory for Computer Science

and

David R. Wilson, National Director
Information Systems Consulting Services
Ernst & Whinney

Evolution of a Model for Computer Integrity

I. Background

More than 18 months ago we presented a model for data integrity in our paper, "A Comparison of Commercial and Military Computer Security Policies," presented at the annual IEEE Symposium on Security and Privacy [Clark and Wilson]. That model, since known as "Clark-Wilson," encouraged the information systems and computer security communities to press forward with integrity-related research. We now wish to give some sense of how the research is going, and, in light of that research, to clarify certain issues raised in our original paper. Those issues involve defining a context for integrity and defining the concept as an aspect of computer security, achieving "real-world" integrity, identifying the features of systems in which integrity is the main security goal, and expanding the U.S. Department of Defense "Orange Book" [DoD] disclosure model to embrace the idea of integrity.

The original paper has generated some follow-on activities. A Workshop on Integrity Policy in Computer Information Systems (WIPCIS) was convened October 27-29, 1987, at Bentley College [WIPCIS]. It was attended by more than fifty researchers and security professionals. A draft of the workshop report was published and distributed at the 1988 IEEE Symposium on Security and Privacy. Three papers were presented at the 1988 symposium describing potential implementations of the Clark-Wilson integrity model [Karger; Lee; Wiseman, et al]. An informal session on the future of the model also was held.

The National Institute of Standards and Technology (NIST) has sustained an interest in the Clark-Wilson Model by releasing the official WIPCIS report and by making integrity security one of its priorities. The NIST has established the Computer and Telecommunications Security (CTS) Council to identify and study key issues and common requirements in the CTS area; a Working Group has been established within the Council to study the area of data integrity. Working Group leader Bob Courtney has recently summarized the results of the group's study.

Other integrity model-related activity includes the recent Canadian Trusted Computer Product Evaluation Criteria Workshop, held in Ottawa, Ontario [Canadian], at which the issues raised by Clark-Wilson were discussed in relation to the U.S. Department of Defense "Orange Book."

II. Context and Definition of Integrity

Because of the precedent set by the United States Department of Defense Trusted Computer System Evaluation Criteria, or "Orange Book" [DoD], many of the implementation schemes for the Clark-Wilson model have focused on computer systems design. This focus has been most necessary and valuable. However, we had intended the Clark-Wilson model as a broader mapping of the issues of integrity that bind real-world concerns to computer system design.

We defined integrity in the original paper as those qualities which give data and systems both internal consistency and a good correspondence to real-world expectations for the systems and data [Clark and Wilson]. Primarily, the expectation of integrity means that systems and data remain predictably constant and change only in highly controlled and structured ways. This concept of integrity is tied to both an internal and an external consistency standard, and is a key element of the Clark-Wilson approach. However, with much work on the subject to date focused more on internal issues, such key concepts as the role of the IVP (Integrity Verification Procedure) and separation of duty have become blurred.

This issue is particularly important because many of the enforcement mechanisms for external consistency require significant internal systems features as part of the basic software and hardware design. For instance, a principle of systems design for separation of duty is that the system must be able to reflect the separation of duty being implemented by application users in real-world environments. This ability to reflect the implemented separation of duty within a system is a complex process which can be greatly simplified if the necessary capabilities are built into the operating environment from the start.

III. Achieving "Real-World" Integrity

By "real-world" we mean the facts, data, and processes outside the computer system which the computer system is expected to reflect, understand, or emulate in some way. Although both internal and external consistency are important, the final test of integrity must be to ensure that the data in the computer is consistent with the world it is intended to represent. If an internal inventory record does not correctly reflect the number of items in stock, it makes little difference if the value of the recorded inventory has been reflected correctly in the company balance sheet.

It stands to reason, then, that integrity controls can never be a matter strictly internal to the computer. A cross-check with the external reality is a central part of integrity control. The computer system can be expected only to preserve the integrity once it has been externally verified.

Methods for Internal Consistency

In our original paper, we described a set of methods for assuring the internal consistency of stored data. This section broadens some of the concepts of internal consistency we introduced in that paper.

Prevention of Change--The simplest method for ensuring the internal consistency of data is to prevent data modification. With this form of control, one need only ensure that the data was correct at one time; since it cannot change it is possible to trust the data from that time forward.

This mode of control is often the one needed within a network. While data is in transit, it is often sufficient to ensure that it does not change at all. Some form of data check function is often used to verify that data has been delivered intact. This form of control becomes less obvious if the network is expected to perform some sort of format conversion of the data, which suggests that reformatting internal to a network is not consistent with this simple form of consistency control.

Attribution of Change--Another important form of control is to bind the data to its author in an unforgeable way. We call this attribution of change, a control which applies to the many sorts of data which do not have a strong internal structure. While accounting records are highly structured internally, an essay on market opportunities is not. With such data, the primary assurance of integrity is the knowledge of authorship, and the assurance that the data has not been modified without the author's knowledge. In this circumstance, a complete log of the data's modification history must be associated with the data, along with the identity of the authors. The system must assure that the data content is exactly that which was provided by the attributed author.

Constraint of Change--For highly structured data such as accounting records, the form of control we call constrained change is applicable. In this mode, the data is modified only by certain programs that have been certified to change the data in constrained ways. We call these programs Transformation Procedures, or TPs [See also Clark and Wilson]. In the example of accounting records, the constraint of double-entry bookkeeping might be enforced: if one account is credited, another must be debited to match.

Partition of Change--The final form of control is partition of change. In this control, the system must ensure that a change is performed by two different people authenticated through user-identifications. Here the system enforces the process whereby no one person has the ability or authority to modify the data and individuals are expected to check each others' work in some manner. The system thus reflects a common business practice, which we describe in our original paper as separation of duty.

In each of the cases given above, the computer system provides controls which enforce internal consistency of the data within the system. These approaches are necessary but not complete. As described earlier,

integrity also requires a correspondence to the outside world. We now discuss three key ways in which a system and data are related to the world they are to represent.

The Integrity Verification Procedure

In our original paper, we introduced the idea of the Integrity Verification Procedure, or IVP. The IVP has a formal relationship to the rest of the model. The proposed proof methodology to demonstrate consistency after running a number of transactions was an inductive one: if each TP takes the system from a valid state to a valid state, then a series of them should take the system through a series of valid states, so the system is finally valid. The necessary condition for this to work is that the system initially be in a valid state. The IVP was proposed to ensure that.

Several people have observed that in a formal sense, this is a redundant feature, as the IVP is just a specific example of a Transformation Procedure, or TP. This observation misses the dual role of the IVP, which is not only to check the internal consistency of the data, but also to verify the consistency between the data and external reality.

Since the IVP checks external as well as internal consistency, it is not just a procedure that is internal to the computer. Instead, it is one of the points where the controls internal to the computer are tied to the larger context of information controls within the organization.

It was observed in one comment that the only reason we need the IVP is that we do not trust the rest of our methodology. Yet this lack of trust does not negate the value of that methodology. Consider again the comparison with a set of accounting records. The books are balanced daily, but once a year, even though good controls have been exercised on normal activities throughout the year, an audit is performed which independently verifies that the records correspond to reality. We need the IVP in the model to capture this idea, accepted in practice, that a system needs a periodic cross-checking.

One other issue associated with the IVP concerns the "reality" a system is measured against. At the WIPCIS Conference there was extensive discussion of integrity domains. When using an IVP to compare a system back to reality, it was recognized that there may be multiple views of that reality depending on the scope of the IVP. These views were defined as integrity domains. It may be necessary, therefore, to label data indicating the particular integrity domain to which it was compared. As systems become large and complex, this comparison with domains will become a necessary process.

The challenge of the IVP is to recognize that those integrity activities that occur outside the computer system must be represented as part of the process of verifying the mechanisms inside a computer whenever possible. There is no way to divorce the outside world from the internal controls on integrity, since integrity is meaningful only in terms of the relation of data to the outside world.

TP Certification

A second major element in assuring the external consistency of data and systems is the TP certification process. This process appears to have two key elements. The first is to assure that the TP does what the specification requests. This includes that source code corresponds to object code, that the TP has been verified and works properly, and that all changes are known and proven as correct.

The second is to assure that the specification for the TP itself corresponds to the "real-world" process it is intended to model. For example, if the TP is to calculate depreciation, the specification should correspond to the correct calculation approach and structure.

When both of these requirements for certification of TPs are taken together, the TP can be assumed to play their part in assuring the integrity of the system and data. These TP certification comments apply both to application TPs and operating system TPs.

Separation of Duty

The separation of duty concepts are the third element confirming that data and systems correspond to the intended real-world model. These concepts have been difficult for everyone to work through and for good reason. Even though they are commonly used in business everyday, they have not been well formalized.

For our purposes, there are several rules concerning separation of duty that are helpful:

1) Adequate separation of duty occurs when the custody of elements of a transaction or assets is so subdivided that no one person can commit significant fraud or error without detection or prevention. For instance, to prevent fraudulent transactions, a person who has custody of assets does not also have custody of the accounting record. Similarly, to avoid error, people who keep subsidiary ledgers do not also keep general ledgers.

2) The best separation of duty occurs when the people involved in the subdivision of responsibilities have substantially different sets of motives and perspectives. Two people performing critical entry-key verification, or two performing the same act to launch a nuclear missile are examples of the weakest form of duty separation. Stronger forms would include an electronic funds transfer where a clerk in the accounting (records) department initiates a transaction, and a supervisor in the treasury (asset) department releases the transaction.

3) The system of controls must be so constructed that significant breakdowns of the system of control can occur only when one or more key elements of the separation of duty are violated through the collusion of the people involved. For example, unauthorized access to the computer center is possible through collusion with

the security officer. Without collusion, such access generally is not possible.

4) The systems of control cannot be violated by the unilateral actions of one person. This rule is implied by Rule 3.

In the original paper, we acknowledged that the implementation of these rules depends upon the specific way a computer system is implemented in a particular setting. But there is enough generality in the ways in which separation of duty is implemented that it is reasonable to expect the operating systems of a computer to have general enough capabilities to reflect almost any particular implementation of separation of duty.

The next section carries many of these principles forward into a features list for systems and computers.

IV. Computer Support of Data Integrity

So far we have identified a number of mechanisms within the computer that provide for data integrity. In this section we gather these features together, and discuss the manner in which they contribute to the overall integrity goal. It is these features, with others that may be identified in future research, that might be incorporated into some evaluation criteria for future computer systems.

Change Logs and Integrity Labels on Data

To establish that separation of duty has been followed, every important access to a system should be tied to a specific person and logged. This means that the author of data should be recorded in an unforgeable way within the data itself, since in many cases the source of the data is the best assurance of the quality of the data. Since data, in general, may undergo a number of modifications as it resides within the system, the record of authorship may need to be a record of the total change history of the object, not just of a single entry. As stated earlier, it may also be necessary to record the integrity status of data by noting the execution of an IVP and the domain used. Some systems currently provide a partial record in the form of two fields, one recording the original author and the other recording the latest author. Application requirements will dictate whether a partial record of this sort is sufficient.

Support of the Access Control Triple

To support the idea introduced above of the constrained change, the system needs to have a mechanism to ensure that data is modified only by selected programs which have been verified in some way to perform only acceptable changes. While TP was intended to capture this idea, the principle of the access control triple is meant to enforce it. The "triple" binds user, program, and data together as a single control object, and thus goes beyond the traditional discretionary control scheme. It may be possible to create the approximate effect of the access control triple by careful

use of traditional access control lists and by representing a program as both subject and object in the permission list, but the result is neither obvious nor precise. For this reason, we believe the access control triple in some form should be a fundamental part of any system oriented towards ensuring the integrity of data.

Enhanced User Authentication

Any system concerned with security in any form must have some means to identify the user to the system. The most common method of achieving these goals is the password. However, if the system is concerned with integrity, either through enforcement of partitioned change or attributed change, there are additional requirements for authentication. The system must ensure that the user's identity cannot be forged and that the identity cannot be shared.

This requirement has not received direct attention in most of the literature on computer security, although the concern applies to disclosure controls as well. But the problem is central to integrity control, especially in the area of separation of duty. If for any reason one user gives away his password to another, then that other user can act inside the system as two people, which may permit him to violate the separation of duty rules.

Since violation of the separation of duty rules is the key to corrupting data and committing fraud, any circumvention of authentication must be viewed with great concern. The problem is that the password system as generally implemented does not itself meet the separation of duty rules. The holder of a password can easily and unilaterally invalidate his own identity by making the password easy to guess, by posting it, or by storing it in his own PC. A means is required to prevent the user, through a unilateral action, from circumventing the authentication mechanism.

Passwords should not be considered a realistic authentication method for a system with high expectations for data integrity. Better methods include challenge-response tests involving a device (called the "token") issued to the user that performs a cryptographic transformation on the challenge. Since the transformation is sealed inside the device, it is only possible to loan one's system identity by loaning the actual device, which is a much less trivial action than telling the password. A more rigorous presentation of these principles, called "see-through security," is provided in an article written in 1986 by Andersen, Clark, and Wilson [Andersen].

Control of Privileged Users

To ensure that the basic protections of the system, such as the access control triple, are not violated it is necessary to regulate strictly the actions of privileged users. Privileged users include those who enter access control triples into the system, register new users, or maintain the operating software. The goal of the controls must be that separation

of duty is not circumvented. For example, people who can add new users to a system should not know the identifier for those users, and should not be able to change the access rules for those users. Similarly, systems programmers who develop software should not be able to install the software.

Application Program Control

Systems concerned with security must ensure that administrative procedures do not corrupt the system's software. For example, a false release tape can be used to insert changes into an operational system. Similarly, replacement of an object module can cause system behavior that cannot be anticipated by review of the sources.

In a system concerned with data integrity, the control to prevent this sort of corruption must be extended to the application programs as well. Such control, however, constitutes a substantial operational burden, as the bulk of the applications code usually swamps the system itself. For this reason, the system should be provided with standard automated aids to manage application software. These should include tools to enforce source and object synchronization, locks to prevent changes to object code without dual controls, logs of changes, and tools to derive flow diagrams from both source and object code that permit understanding of program flow and changes to that flow.

It should be noted that these sorts of tools are easy to postulate, but require significant effort to define and to put into operation. However, this effort is important not only to integrity, but also to good operating practices in general.

Dynamic Separation of Duty Related to TPs

Separation of duty requires that TPs be divided into sets which are executed by different groups of users. In general, a task (for example, purchasing something or writing a check) will be designed as a sequence of TPs rather than as one single TP. By requiring the various TPs in this sequence to be performed by different people, separation of duty is achieved.

The simplest way to achieve this separation is to assign different people to different TPs in a static manner, using the access control triple. The system administrator in charge of maintaining the triples is responsible for understanding the way the rules achieve separation of duty and for assigning TPs to individuals. This works, but is limited in functionality because it is often necessary or desirable to reassign people to tasks dynamically.

An alternative approach is for the system to keep track dynamically of the people who have executed the various TPs in the sequence, and ensure, for any particular execution, that proper separation has occurred. This might better model actual, dynamic requirements in the real world.

There are, however, several hard problems to solve in order to implement this function. First, the sequences of valid TPs must be defined in some way. Next, the allowed patterns of separation must be encoded. Finally, there must be some record in the system of each current execution of a sequence, and each TP being executed must be identified as a part of one sequence. WIPCIS participant Bill Murray has described an application-level implementation of some of these ideas at IBM [Murray].

Finally, there is a possibility some key aspects of separation of duty can be recognized and implemented at the operating systems level. For example, if the operating system were able to recognize that an application is either creating a transaction or approving one, it could automatically require different people to execute these two functions. This concept will require a higher level of operating system intelligence wherein terms like "read," "write," "add," and "delete" are replaced by higher-level terms such as "create," "approve," "review," and "update." In this mode the operating system could enforce at least a primitive type of separation of duty, independent of application controls or even of overt actions taken by a security officer.

V. Evaluation Criteria for Integrity

The previous section outlined a number of features that might be sought in a system oriented toward data integrity. We believe these features should be combined with others used today to provide good support for disclosure control, such as **mandatory** enforcement of the lattice model. The result would be a unified set of evaluation criteria for systems with respect to integrity and disclosure. In this section, we briefly speculate on an integrated set of evaluation criteria, using the Department of Defense "Orange Book" as a starting point.

The problem is to relate the two sets of features, those for integrity and those for disclosure control. In Figure 1 we have listed the various system features mentioned in the paper and proposed a possible (and very speculative) assignment of these features into a three-division rating system. The lowest listed division, "C", would correspond to the Orange Book "C", and would represent a system with only user-discretionary controls for integrity. The "B" level would require a fairly rigorous set of integrity capabilities within the system which--and as much as possible--are required.

Few additional features are added or made to the "A" level. The major issue at "A" is the degree of certification done to prove the functionality described. Throughout, the Orange Book requirements for operating system certification and other control capabilities are expected to be the same. This table shows only additions.

[Figure 1]

A Mapping of Integrity Features
to Existing Orange Book Requirements

Feature	Division		
	A	B	C
Prevention of Change (e.g., a message authentication code)	R/D	R/D	O/D
Data Labels and Logs			
--Change log	R/M	R/M	R/D
--Change log with attribution	R/M	R/D	O/D
--IVP execution log	R/M	R/M	O/D
--Domain logs for data	O/D	O/D	---
Access Control Triple	R/M	R/M	R/D
Application Program Change Control	R/M	R/M	R/D
Uncircumventable User Authentication (e.g., tokens)	R/-	R/-	O/-
Controls on Privileged Users	R/M	R/M	O/D
Dynamic Tracking of Separation of Duties	O/D	O/D	---

M = Mandatory
D = Discretionary

O = Optional
R = Required

Figure 1 shows the optional and required features to support levels "C", "B", and "A". It also reflects our views on those controls which are mandatory versus those which are discretionary. In this case we define mandatory as those controls which are unavoidably imposed by the operating system between user and data. Figure 1 depicts five possible combinations describing degrees of control:

R/M: Control is required for this level and its use is mandatory across all applications.

R/D: Control is required for this level but its use is discretionary (i.e., application-dependent).

O/D: Control is optional at this level and its use is discretionary (i.e., application-dependent).

R/-: Control is required at this level but is not directly related to the operating system being imposed between user and data. (Therefore, there is no mandatory or discretionary stipulation at this level.)

O/-: Control is optional at this level and is not directly related to the operating system being imposed between user and data. (Therefore, there is no mandatory or discretionary stipulation at this level.)

The feature that most distinguishes the integrity model of our original paper is the access control triple. Without this feature, the system cannot effectively enforce constrained changes (our transformation procedures, or TPs), which we believe are the key to a broad class of integrity controls. Support of the triple, we argue, is the key indicator of real support for integrity. The triple becomes mandatory at the "B" level.

Several of the other related features would presumably be included at the "B" division in the criteria. For instance, if access control triples are to be enforced, then the change controls on application programs are needed. Similarly, control is needed for the privileged users who create users and TPs if the triple is to be effective. Thus, there is a consistent set of tools that combine to provide a system which relates to integrity in the same way the lattice model of the Orange Book "B" division relates to disclosure control.

The enhancements proposed above for user authentication also are relevant to systems concerned with disclosure control, and we believe challenge-response authentication could reasonably be factored into any security system, regardless of particular security emphases. We make user authentication tokens required at division "B" because we believe that support for separation of duty is a minimal capability, and that a mechanism stronger than passwords is required for effective separation of duty.

There are several sorts of logging in the table. The simplest is a history log that records the identity of the user of the TP. This level of logging is effective if separation of duty is fixed in a static manner in the access control triples. A more powerful form of logging also records the user associated with each data modification, which provides a more detailed record of responsibility, and also supports that aspect of integrity that is based on the attribution of change.

Another form of logging is to record, for each data item, when IVPs have been executed for that data. This is a variant of a history log which may be separately retrievable, so that a user can determine the last time the integrity of the data was verified.

The most sophisticated form of logging, which remains rather speculative, is the labelling of data to indicate comparison with integrity domains.

The final features are the system support for dynamic partition of TPs to support separation of duty. As was discussed in the previous section, this sort of functionality could be very important if we can invent ways to incorporate it at the operating system level. These features would then be **required** and possibly **mandatory** at the "A" and "B" levels. But since we do not know how to do this, we have indicated in Figure 1 that these features are **optional** and **discretionary**.

VI. Conclusion

We hope this paper clarifies some of the ideas the original paper left undeveloped. Still more work needs to be done in these areas. We believe future research should pay more attention to both the internal and external requirements for integrity. Future research also should focus on the implications of separation of duty, as we have only just started to understand the systems implications of this concept.

The Orange Book has been a very important start for setting industry security standards. Every reasonable attempt should be made to build on its structure. Because of its requirements, many tough problems--such as TP certification--are being tackled successfully. We believe that in the future the difficult problems with making and managing good logs and data labels will need to be addressed as well. Finally, confidentiality and integrity are only two pieces of the computer system security puzzle. The third piece, denial of service, needs to be addressed before we have a really complete approach.

Acknowledgments

The authors wish to thank Steve Lipner of Data Equipment Corporation and Bill Murray of Ernst & Whinney for advice and comments on earlier drafts of this paper. The authors also wish to thank Clark Holtzman of Ernst & Whinney for assistance in revising and editing the paper.

References

Andersen, Robert G., David D. Clark and David R. Wilson. "See-Through Security: A New Approach for Authenticating End Users in an Open Network." MISWeek, 1986.

[Canadian] Proceedings from the Canadian Trusted Computer Product Evaluation Criteria Workshop, August 4-5, 1988, Ottowa, Ontario. Govt. of Canada, 1988.

Clark, David D. and David R. Wilson. "A Comparison of Commercial and Military Security Policies." In Proceedings of the 1987 IEEE Symposium on Security and Privacy, April 27-29, 1987, Oakland, California. Pp. 184-94. Washington, D.C.: Computer Society Press of the IEEE, 1987.

[DoD] Department of Defense Trusted Computer System Evaluation Criteria, CSC-STD-011-83, Department of Defense Computer Security Center, Fort Meade, Md., August, 1983.

Karger, Paul A. "Implementing Commercial Data Integrity with Secure Capabilities." In Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 18-21, 1988, Oakland, California. Pp. 140-46. Washington, D.C.: Computer Society Press of the IEEE, 1988.

Lee, Theodore M.P. "Using Mandatory Integrity to Enforce 'Commercial' Security." In Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 18-21, 1988, Oakland, California. Pp. 130-39. Washington, D.C.: Computer Society Press of the IEEE, 1988.

Murray, William H. "Data Integrity in a Business Data Processing System," Appendix 6. In Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), October 27-29, 1987, Bentley College, Waltham, Massachusetts. Washington, D.C.: National Bureau of Standards, 1988.

[WIPCIS] Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), October 27-29, 1987, Bentley College, Waltham, Massachusetts. Washington, D.C.: National Bureau of Standards, 1988.

Wiseman, Simon, et al. "The Trusted Path between SMITE and the User." In Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 18-21, 1988, Oakland, California. Pp. 147-155. Washington, D.C.: Computer Society Press of the IEEE, 1988.

On Data Quality

A personal comment on the papers by Robert Courtney and Willis Ware

by
Viiveke Fåk
Dept of El. Eng.
Linköping University
S-581 83 Linköping
Sweden

This is a discussion about words, not about actions, equipment, methods etc. I have for years tried to define "data quality" to myself as well as to others. The strawman paper did not make me change my mind on previous conclusions, but it made me see them from a new angle, thus making some points clearer to myself. To start from the very beginning of terminology in computer security:

Security is the not so fortunate term used all over the world for a lot of sometimes competing goals in EDP. In short it means that the correct information should be available to the correct user within the correct time and everything else (incorrect user, incorrect data, incorrect time) is not allowed to happen. Is this data integrity, as it would be if integrity includes quality and quality includes confidentiality? I can not see integrity and security as synonyms. Maybe, if you take integrity to be the state of data and security the state of the system achieving full data integrity. But it does not match too well with either integrity in "non-data" usage or security as used in the data field. Rather it is a matter of two main goals: Confidentiality and integrity.

Unfortunately security can today include both confidentiality and integrity, as when we look at what is included in a computer security conference, and security can refer to just support for confidentiality, as in the Orange Book. We can find this regrettable and somewhat confusing, but there it is and we have lived with it for at least twenty years.

So let us accept the common use of security. Let us also accept the definition of integrity suggested by the integrity working group. I think that it is a good one. Often integrity is taken to mean only that data are not corrupted during their treatment in a computer system. This makes sense only if you add quality as a third criterion.

If we accept the above we obviously have two things:

Integrity can not include confidentiality through data quality. In the real world measures to improve one may even degrade the other.

Data quality must now be defined, or else we have no definition of integrity.

So what is then data quality? Let us first turn to basic words with all their associations and meaning. Quality can mean two entirely different things. To quote Webster: a) an inherent feature, property, characteristic b) degree of excel-

lence, grade. It should be clear that in the integrity definition only b) is meant. To include a) would be to enlarge the meaning of the definition to a point where it looses its meaning. But it is necessary to define what qualities of data should be qualified in order to define data quality.

The characteristics so far mentioned in the US discussion are accuracy, timeliness and completeness. These should obviously be measurable in some sense. But it is obvious that they can not be measured without a description of the ideal that we are comparing them to. Thus data quality divides into two parts: A description of what data we have tried to collect and a statement on how the actual data can be expected to conform to this norm.

Let us for a moment suppose that we have got this description of the ideal. Then we can measure how close registered data are to their description. But to do so presupposes that we can get the accurate value for each field from other sources. In many cases this is true, even though it may be laborious. In these cases it is possible to analyze a small but represantative portion of a larger aggregation of data, register the percentage of errors, and then state that we know on the average the accuracy for those data.

Completeness is not as easy to handle. The description must in some sense define what should be included in such a way that it is also possible to understand what is not included. This will in the end lead us to a discussion of the knowledge of the user, but let us postpone that for a moment. If we have a clear and unambiguous description of what the data aggregation should contain, for example a simple list of what fields should exist in a record, then it is possible to count missing information, for example percentage of fields that are left blank. But we must recognize that in a modern database some fields should have a value only if another field has a specific value. We must also recognize that it may be extremely difficult to find a description that gives a clear indication of what has been left out completely. One example is when all data from a certain district are missing because the district name is missing from the district list. This makes completeness rather complicated to measure.

Timeliness is actually not even directly connected to data as such. It is a function of the system. Deficiencies in the system may result in too old facts in a database. But what do we mean by "too old"? Older than specified in the description? If we are trying to get a quality measure on data as such, this is the only possible definition. It is also useful in many situations. But what kind of description do we need? Age of oldest item? Age of freshest item? Average age? Update frequency? Mean deviation of that frequency? And will it help me to know all these figures, if I do not know the exact age of the data item I am fetching compared to the latest change of that value in the real world? Actually the only possible conclusion is that stored data should contain a declaration about the time when they were accurate. Then it is up to the user to decide if that is what he wants or if he can take the chance that data were accurate also at the time he is really asking about. Can you then imagine the increase of every database, with a "time registered" field for every item!

Still the ground covered so far has been the easy part. We now come to two problems: The user and the data description. They are tightly bound to each other. A user can judge the quality of data only if he can first confirm that the stored data should in an ideal case answer his questions and then learn the

extent to which they can be expected to do so in reality. The latter part is the one discussed above. The first part depends on if he can interpret the data description and if he can interpret the data.

It is quite clear that interpretability depends on the previous knowledge of the user plus further information available in or at the database. How can we measure or indicate that? How can we improve it and know that the result is better? In Sweden the demand for interpretability is included in the definition of data quality without further indications or subdefinitions.

Another basic question: Suppose that a user knows that the contents of a data base are not assembled for his specific purpose, but that they may still answer his questions with some reliability. Or suppose that the description is misleading, so that the user does not know what data he actually is using. This is the highly user oriented aspect of relevance: using correct, timely, complete data for the wrong purpose. How do we treat that?

In short: We need a complete, clear and accurate description of the aggregated data in order to measure how complete, timely and accurate these data are. We must also address the problems of interpretability and relevance.

TERMINOLOGY, CRITERIA AND SYSTEM ARCHITECTURES FOR DATA INTEGRITY

Ravi Sandhu
Department of Computer and Information Science
The Ohio State University, Columbus, Ohio 43210

Abstract. In response to the strawman document [9] we propose that trust be treated as synonymous with integrity rather than synonymous with confidence. We also propose that mandatory controls be taken to mean controls based on properties of the object and/or the subject. Label-based mandatory controls are then a special case of this more general notion. The TCSEC [11] presents criteria for establishing prescribed levels of confidence in trusted systems with particular objectives. We consider how these criteria might be generalized to a broader context. Finally regarding architectures for trusted systems we suggest enhancements to the current security kernel approach.

1 INTRODUCTION

This paper discusses three interrelated topics pertaining to data integrity. In the spirit of this workshop the concepts are not presented as final, definitive or absolute. They do raise many interesting questions which must be confronted, in one form or another; even if the terminology suggested here needs modification and refinement, as it almost surely will.

Our first topic concerns basic terminology for which we have specific proposals regarding "trust" and "mandatory controls." For the most part we agree with the positions argued in the strawman document [9] for this workshop. The document is a significant contribution to our understanding of integrity and lays the foundation for productive debate in future. However we suggest that trust be treated as synonymous with integrity rather than synonymous with confidence. We believe the same arguments used to support a binary view of integrity also apply to the notion of trust. That is trust is a binary property, relative to some context, in whose evaluation we have varying degrees of confidence. The main advantage of our proposal is its explicit recognition that there are two independent issues involved in evaluating trust:

1. What functions is the system trusted to perform or not perform?
2. What is the degree of confidence in our trust?

Regarding mandatory controls we propose the notion be generalized so it is not tied to labels. In our view label-based mandatory controls are a special case of

controls based on properties of the object and/or the subject. In the military non-disclosure context these properties turn out to be best expressed as partially ordered labels, obtained by combining levels and compartments. In other contexts these properties are more naturally obtained in other ways. For instance the type of an object determines what operations can be executed on it. The security community has no handy term for "controls based on properties of the object and/or the subject" although their fundamental importance has often been recognized [3, 5, 18, 24, 26, for instance]. We propose the term mandatory controls be used in this broad sense and that it be qualified when a specific property is intended, such as in label-based mandatory controls.

Our second topic concerns criteria for evaluating trusted systems. The TCSEC [11] recognizes the separation between functionality and confidence noted above when it states, "Included are two distinct sets of requirements: 1) specific security feature requirements; and 2) assurance requirements." However in its criteria this separation is not clearly maintained. In the transition from one class to the next in the C1 through B3 range both functionality and degree of confidence are simultaneously increased. Whereas in the B3 to A1 transition the functionality is unchanged but the assurance requirements are substantially higher. In the classified sector, with its specific objectives for control of documents, there may be a logical joint progression of functionality and degree of confidence. But in a broader context these two issues are best kept separate. Especially in the commercial arena, there is a need for systems with relatively primitive functionality but with high levels of confidence in their evaluation as trusted. For instance the function might be limited to requiring a audit trail which can be trusted with a high degree of confidence. We need finer criteria to enable users to select the combination of functionality and level of confidence in a trusted system that suit their needs within their budgetary constraints.

Our third topic concerns architecture for trusted systems. The conventional approach to security kernels [13] places all trusted code in the kernel and does not trust any code outside the kernel. This approach has been reasonably successful in the non-disclosure context.[1] For a broader context we suggest a multi-layered approach with a table-driven kernel. The kernel is trusted, with a high level of confidence, to enforce the policy specified in its policy tables. These tables are static so they can be placed in read-only memory when the system is "built." Our objective is to separate policy from mechanism to exploit commonality of mechanism across a variety of policies. This kernel can support the "access-control triple" called for by Clark and Wilson [5, 7] as well as separation of duties, perhaps by using transaction control expressions [29]. Above the kernel are trusted layers of application-independent and application-specific code. Code for well-formed transactions, application specific auditing, and policies not directly supported by the kernel tables resides here.

[1] Even in this context there are significant deviations from this ideal. The notion of a trusted subject, which is outside the kernel but must nevertheless be trusted, has slowly crept in. So the architecture is anyway moving in the direction we are suggesting.

2 TERMINOLOGY

We have specific proposals regarding the terms "trust" and "mandatory controls." Our main objective concerning trust is to emphasize two distinct issues in its evaluation, viz., functionality and degree of confidence. This distinction is important in establishing criteria for evaluating trusted systems. Regarding mandatory controls we attempt to generalize the notion so that label-based controls turn out to be a special case. We specifically propose that mandatory controls be used to mean controls based on properties of the object and/or the subject.

2.1 TRUST

The definition of integrity presented in the strawman paper [9] has two salient features.

I. Integrity is a *binary* property. An object either has integrity or it does not.

II. Integrity is *relative* to an a priori expectation of quality in some context. So the same object can have integrity in one context and be devoid of it in another.

The definition appears intuitively sound and useful and provides much needed clarification of terminology. It does leave open the question of what is quality, which is noted as the next task for the Integrity Working Group.

The strawman paper goes on to assert, "Integrity reflects trust in quality. There can, however, be degrees of trust." In appendix I we find the statement of Willis Ware that, "the confidence level in the decision about determination of presence/absence of integrity is indeed continuous in nature and might well be called the "level of trust" in the decision that has been made." The term "degree of trust" is also used in the TCSEC. For instance one of its objectives is stated to be, "to provide users with a yardstick with which to assess the degree of trust that can be placed in computer systems for the secure processing of classified or other sensitive information."

We agree with the spirit of these statements but propose some modifications. The first, and relatively minor, observation is that confidence levels need not be continuous in a mathematical sense.[2] We propose the following statement to allow for different ways of measuring confidence.

III. Confidence in our decision as to whether or not an object has integrity, in some given context, is a concept to which degrees or levels (qualitative or quantitative, continuous or discrete, totally ordered or partially ordered) can be assigned. So it is proper to talk about *degrees or levels of confidence.*

The second, and more important, proposal is that trust and confidence should not be treated as synonymous. Instead trust should be treated as a synonym for

[2]This implication may not be intended but it nevertheless needs clarification.

integrity. Trust is a concept applied for the most part to active agents. It implies an a priori expectation about some aspect of the agent's behavior in a particular context. To be trusted the agent's behavior need only be no worse than we thought it would be. Compare this with the statement [9], "To have integrity it (the quality of an object) need only be no worse than we thought it was." If integrity is treated as a binary attribute, for consistency trust should also be binary. Contrast the following statements.

1. This data has integrity with a high degree of confidence.

2. This data has integrity with a high degree of trust.

The appropriateness of the first statement has been well argued in the strawman paper. We propose the second statement be treated as inadmissible and meaningless, at least in a technical sense. Trust comes in play when we have no choice but to use data in whose integrity we have little confidence. What is being trusted in such cases? We are trusting that all active agents who could have degraded the quality of this data did not do so.

Is this merely hair-splitting? Perhaps, but the proposal is worth investigating if only to spell out its consequences. The proposed viewpoint clearly separates two issues involved in evaluating a system's trust.

1. What functions is the system trusted to perform or not perform?

2. What is the degree of confidence in our trust?

Of course these questions are raised even if trust is viewed as a non-binary attribute. They do become more explicit if trust is defined as a binary property. Moreover there is a recognition that these are really two independent issues.

Our proposal is in direct conflict with the concept of trust in the strawman paper. Consider the following quote from appendix I.

> "... trust is not a binary attribute as the word is commonly used. It is understood that someone (or some thing) is trusted for some purpose (but not others); we are all accustomed to estimating degree of trust in our daily lives. We often say: "Can I trust someone (or some thing) for ..."; thus, we make a value judgment in terms of some end goal, and implicitly the goal has a threat attached to it. This is precisely what trusted systems are all about ..."

It appears inconsistent to us that integrity is a binary property but trust is not. The strawman paper argues that integrity is a binary attribute, which is relative to some context, and in which we legitimately can have varying degrees of confidence. The

examples quoted above show trust is relative, however, they do not show trust is non-binary.

If we accept trust as a synonym for integrity, applicable mostly to active entities, we can specialize the assertions of the strawman paper as follows.

IV. Trust is a *binary* property usually applied to active agents or subsystems which contain one or more active agents. An active entity is either trusted or not.

V. Trust is *relative* to an a priori expectation of quality, particularly quality of behavior, in some context. So the same agent can be trusted in one context and untrusted in another.

VI. It is proper to talk about *degrees or levels of confidence* regarding the decision as to whether or not an agent is trusted, in some given context.

To be concrete consider the following statements where the term process is used in the technical operating systems sense of an executing program.

1. This process has integrity with a high degree of confidence.
2. This process can be trusted with a high degree of confidence.
3. This process has integrity with a high degree of trust.

We propose the first two statements be treated as synonymous. The only difference being that the second statement draws attention to the active nature of a process and quality of its behavior. The third statement we submit should be inadmissible and meaningless, at least in a technical sense. On the other hand, if we equate trust with confidence, statements 1 and 3 above are equivalent while statement 2 can be rephrased as follows.

4. This process can be trusted with a high degree of trust.

Now this is obviously circular and of questionable value in a technical vocabulary. It can only serve to confuse the issue. Our proposal is to treat statements 3 and 4 as inadmissible. The strawman paper in effect takes the position that 2 and 4 are inadmissible. Given a choice between keeping statement 2 or 3 we believe the choice is clearly in favor of 2.

2.2 MANDATORY CONTROLS

The TCSEC draws a sharp distinction between discretionary controls based on identity and mandatory controls based on labels. There appears to be a consensus that a more general notion is needed which is not tied to labels. Consider the following quote from the first WIPCIS report [22].

> "... two types of mandatory controls are considered here — label-based mandatory controls (enforcing separation based on hierarchical or lattice oriented labels, as in the Orange Book) and general mandatory (which lies between label-based mandatory and discretionary controls)."

We are troubled by this characterization of general mandatory as lying between label-based mandatory and discretionary controls. On the contrary we propose that general mandatory be defined so label-based mandatory controls are a special case of whatever we call general mandatory.[3]

A reasonable working definition is given by Clark and Wilson [6] as follows.

> "... the word "mandatory." In the paper, we want to use it in the more general way, to describe any mechanism which is not put into place at the control of the owner of the data, but which is a necessary part of the operation of the system."

However there are situations where mandatory controls are defined by the owner. For example the owner of checks is responsible for defining the well-formed transactions which can operate on checks as well as for defining the separation of duty requirements for processing checks. Once these decisions have been made, at the owner's discretion, the resulting controls are mandatory for all other users. Clark and Wilson also have another working definition [7] as follows.

> "In this case we define mandatory as those controls which are unavoidably imposed by the operating system between user and data."

This is very broad and can be interpreted to include discretionary controls.

We propose to define mandatory controls as controls based on properties of the object and/or the subject. This is as broad and open ended as the above. However it does suggest that one can categorize mandatory controls in terms of the properties on which the controls are based. In the military non-disclosure context these properties turn out to be best expressed as partially ordered labels. In other contexts these properties are more naturally obtained in other ways. For instance the type of an object determines what operations can be executed on that object. Subjects are divided into two classes for this purpose: the type manager who can execute arbitrary operations and all others who can only execute operations exported by the type manager.

Traditional lattice-based controls [2, 10, 17] are obviously a special case of our definition. Discretionary controls are also a special case. Consider the TCSEC definition of discretionary controls as "a means of restricting access to objects based on

[3]It is not surprising the meaning of these terms is still controversial. Many common terms, such as virtual memory, process, fairness, etc., remain controversial even after years of productive use. The goal is not so much to discover the Platonic ideal meaning, but rather to assign meanings which are practically useful, technically consistent and widely accepted (eventually).

the identity of subjects and/or the groups to which they belong." So in this case the property being used is identity and group membership.

We can exclude discretionary controls by refining the definition of mandatory controls to be "controls based on properties of the object and/or the subject (excluding identity of the subject and/or the groups to which it belongs)." In our opinion it is not unreasonable to actually consider discretionary controls as a special case of mandatory controls. We believe the traditional black and white distinction between discretionary and mandatory controls is inappropriate in many contexts. All authority in a system is ultimately obtained by means of somebody's discretionary decisions [20, 21]. The real difference is to what extent discretionary ability can be granted and acquired during the normal operation of a system, and to what extent it gets fixed at system initialization.

Our proposal allows us to categorize mandatory controls along different dimensions. For instance, consider the following progression.

1. Controls based on *identity.* As discussed above this includes discretionary controls.

2. Controls based on *static properties* of the object and subject. These properties are determined at creation and do not change thereafter. Label-based controls of the Bell and LaPadula model [2] with strong tranquillity (i.e., labels are static) are a well-known example. The type based controls of the schematic protection model [26, 28] are a more general example.

3. Controls based on *dynamic properties* of the object and subject. That is the properties on which the controls are based are themselves changeable, presumably in some controlled manner requiring proper authorization. Controls based on the history of an object and the role of a subject, such as enforced by transaction control expressions [29], are one example. Another example is label-based controls without tranquillity [19] (i.e., labels can be modified).

If we assume identity is immutable, 1 is a special case of 2. Similarly, 2 is a special case of 3 if dynamic is interpreted to include static. So there is a logical progression.

Group membership does not figure in the above categorization. This is deliberate. If group membership is a static attribute we could include it in under 2. However the moment group membership is dynamic a new set of questions is raised [25]. For example consider the following policy.

1. A project group must have a majority of members from within the department.

2. Any department member can unilaterally join any project group.

3. An outsider can be enrolled in a group only by a project supervisor.

This is by no means a complicated policy. Yet there are no systems today which can conveniently support it. A C2 system is not good enough since it cannot enforce the specified mandatory controls. Neither do the labels of B or A systems help.

Another categorization might consider the nature of these properties along a different dimension, for instance as follows.

1. Controls based on properties of an object which *depend on the value* of the data it contains.

2. Controls based on properties of an object which are *independent of the value* of the data it contains.

Such distinctions are important for two reasons. First we can conclude that certain kinds of controls are needed to achieve particular objectives. For instance dynamic separation of duties appears to require controls based on the history of an object whereas static separation can be achieved by controls based on static properties. This gives us guidelines regarding what features are required to achieve our objectives. Secondly we can design operating system mechanisms with the fundamental nature of the controls in mind rather than considering specific applications.

3 CRITERIA

We now turn to consideration of criteria for evaluating trusted systems. There are two major points we wish to make in light of the preceding discussion. Firstly the criteria must clearly separate the issues of functionality and degree of confidence. Secondly we need a finer grain of functionality than provided in the TCSEC.

Separation between functionality and confidence is noted in the TCSEC in its statement, "Included are two distinct sets of requirements: 1) specific security feature requirements; and 2) assurance requirements." However in the TCSEC classes this separation is not clearly maintained. In moving up to higher classes (e.g., from C1 to C2) there is an increase in functionality as well as an increase in the level of confidence required. The sole exception is the B3 to A1 transition which does not introduce additional functionality but considerably increases the required degree of confidence. Clark and Wilson [7] have tentatively proposed a similar progression for integrity evaluation criteria. Since their objective was to arrive at an integrated set of criteria for evaluation for integrity and non-disclosure it is natural that their proposal mirrors the TCSEC. Specifically they propose three divisions with the C to B transition requiring a simultaneous increase in functionality and assurance while the B to A transition is mostly concerned with assurance.

While some joint progression is inevitable we feel it is inappropriate to couple these two issues too tightly. Especially in the commercial arena, there is a need for systems with relatively primitive functionality but with high levels of confidence in

their evaluation as trusted. For instance the function might be limited to requiring a audit trail which can be trusted with a high degree of confidence. This makes breach of trusted behavior by individual users detectable so threat of punitive action may be enough to prevent it.

Moreover the progression of functionality defined in the criteria should be finer grained. This is especially so if an integrated set of criteria are proposed. For instance consider a user who requires high confidence in authentication but is willing to accept low confidence for non-disclosure. In the TCSEC the trusted path required for the former is coupled with the requirement of covert channel analysis. We view functionality as inherently multi-dimensional. For instance consider the following progression of functionality of mandatory controls.

Class	Mandatory controls based on
a	Dynamic properties including value
b	Limited dynamic properties (e.g., transaction control expressions)
c	Almost static properties (e.g., weak typing, labels without tranquillity)
d	Static properties (e.g., strong typing, labels with tranquillity)
e	Identity based (i.e., only discretionary controls)

We do not think it proper to require that a system which includes features of class a must also include all features of the classes below a. The progression is multidimensional. For instance we may require sophisticated support for hierarchical groups [27] and some little amount of controls from the other classes. As a user one would like a rating which measures the features provided in each one of these classes so one can shop for the best match. Levels of confidence might be assigned in a strictly increasing sequence, for instance as follows.

Class	Level of confidence
a	Formally verified
b	Informally verified
c	Extensive testing
d	Minimal confidence
e	No confidence

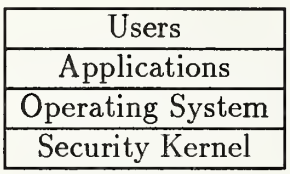
However if we have multi-dimension functionality we expect a different level of confidence to be attached to each dimension to give a multi-dimensional rating.

As a general principle of system design a user should be able pay for the functionality and level of confidence in a trusted system that suits his needs. He should be able to trade one for the other to meet budgetary constraints. The marketplace and technology will determine what combinations of functionality and confidence get supported and at what price. With proper modular designs, system vendors should be able to support large numbers of combinations. It should eventually be possible to upgrade from one package of functionality and confidence to a superior one.

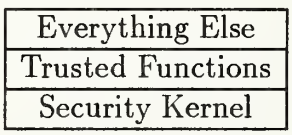
We need finer criteria to enable users to select the combination of functionality and level of confidence in a trusted system that suit their needs within their budgetary constraints.

4 SYSTEM ARCHITECTURES

Our third topic concerns architecture for trusted systems. The conventional approach to security kernels [1, 13, 16] places all the trusted code in the kernel and does not trust any code outside the kernel. The ideal picture is given somewhat as follows.



In practise the ideal is often violated by placing some trusted code outside the security kernel. This gives us the following view.



There are two major reasons why we need trusted functions, or trusted processes, outside the security kernel.

1. Trusted processes need to bypass mandatory controls of the security kernel in order to achieve their objective. For example a downgrader must selectively violate confinement.

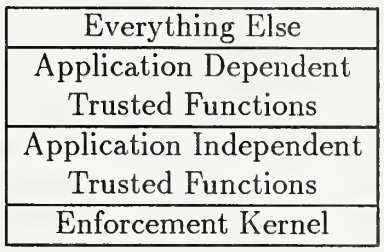
2. Trusted processes perform functions which are security related, so we need a high level of confidence in their correctness, but these do not need to violate confinement. For example a labeler or a backup utility.

Trusted functions are clearly part of the overall security component of a system. Trusted functions which are privileged to bypass kernel controls are particularly disturbing. By their very nature, it is difficult to come up with a rigorous definition of what these processes are supposed to do. Without a precise specification it becomes somewhat pointless to try and verify them.

It is possible for enforcement mechanisms in the security kernel to help us increase the level of confidence in these trusted functions. For instance SCOMP [12] uses "integrity labels" for this purpose while SAT [4] provides a type enforcement mechanism. Of course, neither of these can guarantee the correctness of trusted functions. The

controls increase our confidence by making it more difficult to plant Trojan Horses in trusted code as well by limiting the damage that might be done. The SAT approach is particularly flexible and attractive.

In view of this experience we propose the following idealized architecture for trusted systems.



For the moment let us ignore the non-disclosure problem. What then might one expect in these layers? We propose the enforcement kernel implement mandatory controls which are for the most part based on static properties of subjects and objects. To separate policy from mechanism the kernel should be table-driven. This is by no means a new idea [4, 8, 23] and has obvious appeal. The separation of trusted functions outside the kernel, into application independent and application dependent, is intended to encourage reuse of trusted functions across applications.

We propose the mandatory policy of the kernel be defined in terms of types of subjects and objects, and that the kernel enforce strong typing (i.e., the type of a subject or object is determined when it is created and thereafter does not change). Type-based controls are surprisingly powerful. They gives us the basis for enforcing the "access control triple" of Clark and Wilson [5, 7]. They also provide enforcement of, even dynamic, separation of duties by means of transaction-control expressions [29] or some similar mechanism. There is also evidence that policies based on types are easier to analyze for safety [26] as compared with policies specified without a built-in notion of types [14, 15].

How does non-disclosure fit into this picture? There are several approaches one might take. If labels are regarded as a special case of types the enforcement kernel can handle label-based controls. This has been formally demonstrated in [28]. Of course the covert channel problem remains. So, to achieve B2 or higher ratings, the enforcement kernel and policy tables must be scrutinized for covert channels. Another approach could be to run the enforcement kernel above a traditional security kernel. Such a security kernel could be stripped of all features not related to label-based confinement and conceivably could be small enough to meet A1 criteria. The enforcement kernel could then be viewed as an integrity kernel sitting above the security kernel. Since projects such as SAT are proposing similar features it may be better to split them vertically into two layers.

5 CONCLUSION

To summarize we have considered several topics relating to data integrity.

1. In response to the strawman document [9] we propose that trust should be viewed as a synonym for integrity used mostly to describe active agents or systems containing such agents. This serves to emphasize two distinct issues in evaluating trust, viz., functionality and degree of confidence.

2. We propose that mandatory controls be used to mean controls based on properties of the object and/or the subject. Label-based controls are then a special case of this more general notion. If we choose we can exclude discretionary controls by excluding properties based on identity of the subject and/or the groups to which it belongs. Otherwise we can view discretionary controls as a very special case of mandatory controls.

3. With the above perspective we have argued that criteria for evaluating trusted systems must clearly separate the issues of functionality and degree of confidence. While some joint progression may be inevitable we feel it is inappropriate to couple these two issues too tightly. We have also argued that the criteria need to be finer than those presented in the TCSEC [11].

4. Finally we have considered the security kernel approach to trusted systems architecture. Experience indicates trusted code is needed outside the kernel [13]. This is even more so for application specific integrity policies. The architecture should support a distinct layer above the kernel in which this code resides. Policy and mechanism should be separated in the kernel. Policy should be specified by static policy tables which can only be changed, under careful control, when the system is built. A built-in notion of types should be provided for specifying these policies.

In the spirit of this workshop these proposals are somewhat speculative and need further work. Many of the questions we have raised must eventually be confronted, in one form or another, even if the terminology in which they are phrased is modified.

It appears to us that mandatory controls for integrity will be an order of magnitude more complex than label-based mandatory controls for non-disclosure, in all respects except for the formidable covert channel problem. This is the distinguishing characteristic which makes non-disclosure such a difficult problem. As a final note we express our support for the following viewpoint [20].

> "Many people have assumed that security policies for commercial systems are either less friendly versions of academic policies or military policies with fewer teeth. Neither is true. Considerations for commercial security policy differ in quality, because the principles of internal control take a much more sophisticated view of authority."

References

[1] Ames, S.R., Gasser, M. and Schell, R.R. "Security Kernel Design and Implementation: An Introduction." *Computer* 16(7):14-22 (1983).

[2] Bell, D.E. and LaPadula, L.J. "Secure Computer Systems: Unified Exposition and Multics Interpretation." MTR-2997, Mitre, Bedford, Mass. (1975).

[3] Boebert, W.E. and Kain, R.Y. "A Practical Alternative to Hierarchical Integrity Policies." *8th National Computer Security Conference*, 18-27 (1985).

[4] Boebert, W.E., Kain, R.Y., Young, W.D. and Hansohn, S.A. "Secure Ada Target: Issues, System Design, and Verification." *8th National Computer Security Conference*, 18-27 (1985).

[5] Clark, D.D. and Wilson, D.R. "A Comparison of Commercial and Military Computer Security Policies." *IEEE Symposium on Security and Privacy*, 184-194 (1987).

[6] Clark, D.D. and Wilson, D.R. "Comments on the Integrity Model." In [30].

[7] Clark, D.D. and Wilson, D.R. "Evolution of a Model for Computer Integrity." These proceedings.

[8] Cohen, E. and Jefferson, D. "Protection in the Hydra Operating System." *5th ACM Symposium on Operating Systems Principles*, 141-160 (1975).

[9] Courtney, R.H. "Some Informal Comments About Integrity and the Integrity Workshop." These proceedings.

[10] Denning, D.E. "A Lattice Model of Secure Information Flow." *Communications of ACM* 19(5):236-243 (1976).

[11] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria.* DoD 5200.28-STD, (1985).

[12] Fraim, L.J. "Scomp: A Solution to the Multilevel Security Problem." *Computer* 16(7):26-34 (1983).

[13] Gasser, M. *Building a Secure Computer System.* Van Nostrand Reinhold (1988).

[14] Harrison, M.H., Ruzzo, W.L. and Ullman, J.D. "Protection in Operating Systems." *Communications of ACM* 19(8):461-471 (1976).

[15] Harrison, M.H. and Ruzzo, W.L. "Monotonic Protection Systems." In DeMillo, R.A., Dobkin, D.P., Jones, A.K. and Lipton, R.J. (Editors). *Foundations of Secure Computations.* Academic Press (1978).

[16] Landwehr, C.E. "The Best Available Technologies for Computer Security." *Computer* 16(7):86-100 (1983).

[17] Lee, T.M.P. "Using Mandatory Integrity to Enforce "Commercial" Security." *IEEE Symposium on Security and Privacy*, 140-146 (1988).

[18] Linden, T.A. "Operating System Structures to Support Security and Reliable Software." *ACM Computing Surveys* 8(4):409-445 (1976).

[19] McLean, J. "The Algebra of Security." *IEEE Symposium on Security and Privacy*, 2-7 (1988).

[20] Moffett, J.D. and Sloman, M.S. "The Source of Authority for Commercial Access Control." *IEEE Computer* 21(2):59-69 (1988).

[21] Murray, W. H. "On the Use of Mandatory." Position paper in [30].

[22] Parker, D.B. and Neumann, P.G. "A Summary and Interpretation of the Invitational Workshop on Integrity Policy in Computer Information Systems." In [30].

[23] Popek, G.J. and Farber, D.A. "A Model for Verification of Data Security in Operating Systems." *Communications of ACM* 21(9):737-749 (1978).

[24] Saltzer, J.H. and Schroeder, M.D. "The Protection of Information in Computer Systems." *Proceedings of IEEE* 63(9):1278-1308 (1975).

[25] Sandhu, R.S. and Share, M.E. "Some Owner Based Schemes with Dynamic Groups in the Schematic Protection Model." *IEEE Symposium on Security and Privacy*, 61-70 (1986).

[26] Sandhu, R.S. "The Schematic Protection Model: Its Definition and Analysis for Acyclic Attenuating Schemes." *Journal of ACM* 35(2):404-432 (1988).

[27] Sandhu, R.S. "The NTree: A Two Dimension Partial Order for Protection Groups." *ACM Transactions on Computer Systems* 6(2):197-222 (1988).

[28] Sandhu, R.S. "Expressive Power of the Schematic Protection Model." *Computer Security Foundations Workshop*, 188-193 (1988).

[29] Sandhu, R.S. "Transaction Control Expressions for Separation of Duties." *4th Aerospace Computer Security Applications Conference*, 282-286 (1988).

[30] Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), Bentley College, MA, October 1987, NIST.

INTEGRITY CONTROLS FOR MILITARY AND COMMERCIAL APPLICATIONS, II

Robert R. Jueneman

Presented at:

Invitational Workshop on Data Integrity
National Institute of Standards and Technology
25-27 January 1989, Gaithersburg, MD

By:

Computer Sciences Corporation
Special Projects Division
3160 Fairview Park Drive
Falls Church, VA 22042
703/237-2000

Revised: 1 March 1989

Table of Contents

INTEGRITY CONTROLS FOR MILITARY AND COMMERCIAL APPLICATIONS, II[1]

Robert R. Jueneman
Computer Sciences Corporation
3160 Fairview Park Drive
Falls Church, VA 22042
703/237-2000

1 ABSTRACT

A system of integrity controls is presented that is intended to be suitable for both commercial and military applications involving arbitrary unsecure networks such as ISDN. The integrity threats that are present in a large and dynamic network of TCBs are examined, and a set of rules that are consistent with those of Clark and Wilson are proposed to address those threats. A definition of integrity is provided and an integrity metric proposed, consisting of a non-hierarchical component — a set of *integrity categories* corresponding to a set of integrity domains, *i.e.*, a collection of application-specific syntactic and semantic rules which apply to any objects read and/or written by a process; plus a *hierarchical integrity component* — the estimated probability that all of the rules of the various integrity categories are obeyed. Cryptographic checksums and digital signatures are used to detect and eventually recover from the unauthorized modification of data, instead of attempting to prevent such changes that may occur outside of a TCB or trusted network. It is concluded that a system of Mandatory Integrity Controls consisting of the Biba hierarchical integrity policy with integrity categories and multilevel integrity-trusted subjects, plus Discretionary Integrity Controls which allow a subject to specify which objects are to be accepted or believed (based on the identity of the originator of the object), implemented on a TCB that meets the TCSEC requirements for B2 with respect to security, can provide integrity controls that are well-suited to the networking environment.

2 INTRODUCTION

The present version of the Trusted Computer Security Evaluation Criteria (TCSEC), DoD 5200.28-STD, commonly called the "Orange Book," is almost exclusively concerned with preventing information compromise, *i.e.*, the release of information to unauthorized users. Although preventing information compromise is vitally important to many applications in both the military and the civilian environment, there is an increasing recognition that the TCSEC is seriously deficient in not addressing the concerns of both the Command and Control community within the Department of Defense and the "commercial" data processing requirements within the Government and the civilian sector. The Trusted Network Interpretation of the

1. This is a revised and extended version of "Integrity Controls for Military and Commercial Applications," **Fourth Aerospace Computer Security Applications Conference**, Orlando, Florida, December 12-16, 1988; Computer Society of the IEEE, Washington, DC, 1988, pp 298-322.

TCSEC, NCSC-TG-005 Version 1 (the TNI, or "Red Book"), addresses the need for additional security services for communications integrity (including authentication, communications field integrity, and non-repudiation of origin and delivery), but these services are not sufficient in and of themselves to satisfy the needs of "commercial" users.

The differences between the concerns of the TCSEC for preventing the disclosure of information to unauthorized users (representing the "military" requirements) and the accounting controls usually associated with the control of errors and the prevention of fraud (representing the "commercial" requirements, but equally applicable to the Command and Control environment[2]) were highlighted in a landmark paper by Clark and Wilson[3] at the IEEE Security and Privacy Symposium in Oakland in April, 1987. Since it was presented there has been much discussion about the nature of integrity and how best to control it, but the general principles (if not all of the mechanisms and conclusions) of Clark-Wilson have been widely recognized.

However, the implicit context of Clark and Wilson's observations was that of a monolithic mainframe computer processing the traditional MIS programs — accounts receivable and payable, inventory control, payroll, etc. Little or no attention was paid to such critical activities such as computer program development, CAD/CAM, robotics, nuclear reactor control, hospital patient monitoring, air traffic control, embedded tactical applications, the Strategic Defense Initiative, or even simple word processing. In addition, there was no discussion of the requirement for Electronic Data Interchange of purchase requisitions, invoices, or contracts between disparate organizations which would necessitate the use of digital signatures that would stand up in court if challenged.

The requirements for integrity are relatively independent of the hardware platform that is used, whether we are talking about a small personal computer that is used by a student in college, or a large supercomputer used to design nuclear weapons. In fact, given the incidence of computer viruses, the degree to which modems and LANs are used, the frequent use of public-domain software with personal computers, and the lack of integrity controls provided with the available operating systems, both the threat and the vulnerability may be greatest in the PC environment. Certainly the resources available per user to solve the problem are minimal. On the other hand, the tremendous number of personal computers and workstations that have been sold, together with the hardware architecture of the Intel 80286 and 80386 and the Motorola 68030 chips, may make it easier to solve the integrity problem on PCs and workstations than on the more traditional mainframe and minicomputers. For that reason, to the extent that there is an unconscious bias in this paper, it will reflect a single user, dedicated workstation environment.

In general, if there were no need for communication and if users wrote all of their own programs, there would be relatively few integrity concerns in the single user workstation environment except for those caused by the user's human errors. Locking the unit to prevent access to the hard disk and locking floppy disks in a drawer to prevent theft and/or malicious attempts to change the data contents would go a long way towards providing adequate security and integrity, at least if potentially erroneous or even outright malicious programs and/or data were never brought in from the outside world. However, in both the commercial and military environment, the trend is toward greater and greater degrees of connectivity, through LANs, dial-up telephone links, packet-switching networks, electronic mail systems, public domain bulletin boards, and even hand-to-hand floppy disk interchange. In this environment, where users share data and programs and multiple

---

2. The concept of mission-critical data is discussed at some length in Air Force Regulation AFR 205-16. In that document the treatment of criticality includes both the general notion of integrity and protection against denial of service that would prevent the system from providing the intended service for an amount of time that would impact the mission adversely. Missions are categorized as HIGHLY-CRITICAL, CRITICAL, and NONCRITICAL, but there is no specific guidance provided that would indicate how to build a system that would guarantee that the requirements for a CRITICAL or HIGHLY-CRITICAL mission will be satisfied.

3. Clark, David D. and David R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," **Proceedings of the IEEE Symposium on Security and Privacy**, April 27-29, 1987, Oakland, CA; Computer Society Press of the IEEE, Washington, D.C., 1987, pp 184-194.

individuals have the ability to modify other user's data, either accidentally or deliberately, the old physical isolation controls are no longer adequate.

This paper will therefore address the integrity requirements of very large and dynamic *networks* of interconnected Trusted Computing Bases (TCBs) which communicate over a virtual network of arbitrary non-secure media.

Cryptographic and other mechanisms will then be proposed which would satisfy those requirements. A similar approach based on layered encryption can be taken to satisfy the *secrecy* requirements of interconnected TCBs without imposing the requirement that the entire network be evaluated and trusted, but that would exceed the scope of this discussion.[4]

We will assume that the TCBs provide the equivalent of at least a B2 level of trust in accordance with the TCSEC, so that mandatory and discretionary access controls and labels are provided together with the process isolation achieved through the provision of distinct memory spaces under the control of the TCB, plus the use of memory segmentation and read/write protection. In addition, in order to support the concept of separation of duty, we will require the ability to *exclude* individuals or groups of individuals from accessing specified objects — a feature of B3/A1 systems. Finally, in addition to the traditional controls of a stand-alone, multilevel-secure TCB, we will assume that the TCB is provided with an embedded cryptographic functionality which can be used to protect both the secrecy and integrity of data stored locally or exchanged via the network.

For purposes of discussion, it will generally be assumed that the objects to be protected are files that may or may not be securely installed within the security perimeter of a TCB. They may be stored on a network file server or local hard disk that is not necessarily secure, or they may have been transmitted or transported over unsecure media. For that reason, it will be assumed that the files will be encrypted if necessary to prevent their disclosure to unauthorized individuals and that the mandatory and discretionary access control requirements of the TCSEC are implemented through a system of access control labels and personnel clearances.

The details of the particular cryptographic algorithm(s) and the key management scheme(s) to be used, although highly important to the implementation, are not relevant to the discussion to be presented below. However, it will be assumed that three specific capabilities are provided:

1. An encryption algorithm must be provided to protect the secrecy or privacy of information. The Data Encryption Standard (DES) algorithm would satisfy this requirement for unclassified data.

2. A cryptographic checksum algorithm must provide at least a 128-bit checksum of a file, record, or field, to be used in verifying that the information so protected has not been changed since it was created. This checksum may either be a Message Authentication Code (MAC) which makes use of a secret cryptographic key, or it may use a Manipulation Detection Code (MDC) which is independently protected by encryption or a digital signature mechanism.[5]

3. A digital signature mechanism must be available to provide non-repudiation protection of both the authorship of a document and the document's contents. Such a mechanism may also be very useful in key management. The Rivest-Shamir-Adleman (RSA) public-key

---

4. Jueneman, R. R., "End-To-End Compromise and Integrity Controls," Computer Sciences Corp. Technical Report CSC-TR/87-3001, May 1987.

5. Jueneman, R. R., "Electronic Document Authentication," *IEEE Network*, vol. 1, no. 2, April 1987, pp 17-23. Note however that Dr. Donald Coppersmith has recently discovered an attack against the specific MDC algorithm (QCMDCV4) discussed in that paper which reduces the work factor necessary to defeat it to approximately 10^{11} random trials. Although it would require a non-trivial amount of work to sort or compare all of those trials, it could be done. Unfortunately, as discussed in the paper, the ANSI standard X9.9 Message Authentication Code suffers from a number of deficiencies, and can be defeated by using approximately 65536 random trials for the usual 32-bit version, or approximately 10^9 random trial for the full 64-bit version, and is therefore not suitable for general use. As a result, at the present time I am not aware of *any* published cryptographic checksum algorithm which is completely satisfactory. The rest of this paper will assume that this situation will somehow be remedied.

algorithm is the best known example of a cryptographic mechanism that would satisfy both the digital signature and key management requirements.

Although it would certainly be desirable for the embedded cryptographic capability to be implemented in hardware, both for reasons of enhanced security and for speed, there is no inherent reason why hardware encryption must be used so long as the encryption, checksum, and digital signature mechanisms are contained within the security perimeter of a Trusted Computing Base (TCB). Implementations of DES in software are fast enough for many applications, and although the RSA algorithm is quite computationally intensive for longer keys, substantial strength can be achieved with practical running times with software implementations.

3 THREATS AND COUNTERMEASURES

Clark and Wilson's paper proposed a number of integrity "rules" which attempted to summarize and synthesize what was stated to be standard commercial practice, and certainly an enormous amount of credit must be given to the authors for their efforts. However, during the detailed examination of the Clark-Wilson model that was conducted at the WIPCIS conference[6] it became clear that the business practices that were being abstracted were primarily the traditional double-entry bookkeeping systems as implemented on conventional mainframe computers, and that many of the concerns of more distributed processing systems were not adequately addressed.

This section will analyze the various threats to integrity that exist in a *distributed network of TCBs*, and present a set of rules in the spirit of Clark-Wilson which are necessary to contain those integrity threats. Section 4 will then propose a set of integrity policies and implementation mechanisms which satisfy those rules.

3.1 Integrity Threats

The TCSEC requires that the TCB itself be developed, inspected, and tested in accordance with demanding criteria (especially at the A1 level) in order to ensure that the TCB will work correctly. However, those same criteria are not required to be applied to or enforced upon application programs and/or data controlled by the TCB. In particular, the TCSEC does not require a TCB to implement those mechanisms that would be suitable to enforce the fundamental principle of *duly authorized changes* to applications programs and/or data. The TCSEC requires that B2 and higher evaluation class systems implement the principle of "least privilege," but that only applies to the TCB code and relevant data (*e.g.*, tables), and not to application programs and data. In addition the TCSEC only requires that strict configuration control and a trusted distribution mechanism be applied to the security-relevant portions *of the TCB itself*, and then only for class A1 TCBs.

As a result, the integrity of a data processing system could potentially be compromised by permitting high integrity processes to read low integrity (corrupted) data or alternately by allowing low integrity processes to write or modify high integrity data, where data also includes programs. These potential integrity compromises include:

- Active modification attacks against data by an external adversary during transmission or storage of the information outside of the TCB.
- Accidental modification of data within or outside the TCB during transmission or while in storage, either by random hardware errors or by program "bugs."
- Modification, deletion, or renaming of one user's data by another user, either accidentally or deliberately, but without authorization.

6. **Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)**, Bentley College, Waltham, Mass. October 27-29, 1987. NIST Special Publication 500-160, GPO, 1989.

- Modification of data within the TCB by Trojan horse programs and/or computer virus attacks. Such malicious programs may do direct and immediate damage to *data* (either by destroying the data, or perhaps worse, corrupting the data with insidious changes to the data values), or they may modify other *programs*, perhaps by infecting those programs with a copy of themselves.

Loss of accountability for programs and data could occur, including changes to programs and data by unauthorized personnel, in addition to loss of control through the execution of programs in unauthorized circumstances,[7] as follows:

- Fraud and/or the abuse of authority, including unauthorized inputs or program execution, the violation of the principle of separation of duty, and the attempted repudiation of responsibility for commitments made or actions taken.
- Failure of containment, wherein valid programs are executed in invalid ways, *i.e.*, run twice, run out of sequence, run at night or on weekends, etc.
- Failure of control, through the admission of incorrect data into the system or the inability to take corrective or restorative action in the event of an error or failure through the use of normal transaction journals, backup files, etc.

Finally, we must not ignore the problem of ensuring the continuing correspondence between the computer databases (a model of the external world) and the external world itself in many applications. In addition, there may be a need for a periodic comparison or reconciliation of two or more databases within the system (*e.g.*, a primary and a backup copy of a database that are supposed to be maintained in synchronism with each other in quasi-realtime on two physically separated machines, despite possible hardware, software, communications, and procedural errors) in order to detect and/or correct any discrepancies that may occur.

Clark and Wilson rightly point out the need for what they call an Initial Verification Procedure (IVP) in those instances. The IVP not only screens the initial input data to the system, but it also represents a checkpoint with reality. Even if the computer system operates absolutely flawlessly (unlikely, for systems of meaningful size), and even if the human beings who input the data to the programs never make a mistake (even less likely), in the case of an inventory control system, for example, it is still necessary to periodically go out to the storeroom and count the items on the shelves, if only to detect the possibility of theft. The periodic reverification of the internal consistency of the computer model and of the correspondence of the computer model with the external "real world" via the IVP is therefore an absolutely essential element of *closed loop control* of the overall system. Without this "balancing of the books," the model may slowly lose accuracy as errors begin to accumulate both inside and outside.

However, the problem of correspondence between the model and reality is primarily one of correcting human mistakes and compensating for internal errors and external events (and sometimes the design of the model). Other than being aware of the requirement, the design and application of the IVP will not concern us greatly here. Indeed, in some cases there *is* no external world to model, yet we can still talk meaningfully about the integrity of a contract, or of the accuracy of translation of poetry from one language into another, or even the correctness of a presentation on a computer screen of an abstract visual image which is only defined by a computer process.

For that reason we will focus primarily on the threat of accidental errors being introduced during the storage, transmission, or transformation of that information, and the possibility of deliberate errors being introduced by external agents or through the malfeasance of authorized users.

---

7. Cf. Clark and Wilson, *ibid.*, for further discussion and examples.

3.2 Viruses and Trojan Horses

Within certain segments of the Government, and despite the lessons taught by the Walker family and other recent spy cases, there is a point of view that argues that since their people are all highly cleared and presumably trustworthy, they have no need for high levels of computer security assurance. It is not surprising, therefore, that many responsible individuals in the commercial sector have also asserted that they have no need for high levels of computer security functionality or assurance. This is not necessarily because the civil sector trusts all of their users, but because in general commercial organizations have not instituted a rigorous policy of clearing or screening their personnel as to who can have access to what information, and they therefore do not have a reliable basis on which to implement mandatory access controls.

Unfortunately, opinions such as these miss the point rather badly, both from the standpoint of security and the less well-recognized threats to integrity and availability of service. **It is not only untrusted users who are a threat, but also the untrusted *programs* which run on behalf of trusted users that can cause grave damage.**

At the lower levels of computer security assurance (B2 and below), there is no requirement to carefully partition the operating system into security-critical and non-critical portions and to minimize the amount of security-critical code. As a result, virtually all of the code in the operating system has to be trusted, and the sheer size of most operating systems makes it almost impossible to be sure that all possible loopholes and security weaknesses have been closed. As Shockley[8] has pointed out, "unless the TCB is small enough to be thoroughly analyzed and tested by restricting its contents to security-critical modules only — *i.e.*, by basing the system on a Class B3 or A1 TCB with a security kernel" — it is possible that a specific weakness exists that could be attacked by a virus or Trojan horse program, and the TCB itself could be subverted. And although a trusted distribution mechanism and strict configuration management is required at the A1 level, if the tools that are used to construct the TCB were ever infected with a virus, even the security-critical modules of an A1 TCB could be jeopardized.

In Shockley's scenario, the attacker might target an authorized user of the system by planting a seemingly benign program on a bulletin board that the user was known to access, hoping that he would pick up that program and run it on the classified machine, thereby infecting it. The virus-containing program could later be deleted from the classified system, but the damage would have already been done. The modified portion of the operating system might then do nothing but to lie quietly in wait for an external signal, at which time it would compromise all of the data on the system to a confederate on the outside.

Although the thought of a virus allowing all of the order-of-battle plans to be compromised just before an important engagement is to be fought is bad enough, it is important to realize that the TCSEC does not require even an A1 TCB to provide any mandatory controls to protect those same order-of-battle plans from being *modified* or *destroyed*. In particular, if a user executes a Trojan horse program, that program inherits all of the attributes of the user, so it could surreptitiously modify or destroy any files that user would have access to, thereby preventing any battle planning at all. In the commercial sector, a virus might wipe out all of the files on the system just before the books had to be closed at the end of the year, for example. Significant disruption could be caused, ranging from the payroll not being met to a brokerage firm not being able to settle stock trades on time, with disastrous results.

The recent publicity given to the efforts necessary to eradicate the recent Internet computer virus or worm, the "Christmas" virus that caused the IBM's corporate electronic mail system to be shut down in December 1987, and the spate of attacks against PCs should convincingly illustrate how vulnerable to virus and Trojan horse attacks almost all systems are at present, and why some additional form of protection is urgently required.

---

8. Shockley, W. R., R. R. Schell, and M. F. Thompson, "The Importance of High Assurance Computers for Command, Control, Communications, and Intelligence Systems," **Fourth Aerospace Computer Security Applications Conference**, Orlando, FL, December 12-16, 1988, Computer Society Press of the IEEE, Washington, DC, 1988, pp 331-342.

With regard to the threat of computer viruses and Trojan horse programs, Cohen[9] has pointed out that it is theoretically impossible to guarantee the detection of a virus by its appearance alone, since whatever detection mechanism is proposed to determine whether a particular program will infect another program could be used in a contradictory sense by another virus. Viruses could also evolve or mutate, so they could avoid whatever antiviral detection schemes are introduced and in the process introduce "super-infections," where a program may be infected by more than one virus.

In addition, it is difficult or impossible to detect a Trojan horse program by its behavior, since it may use services which are legitimate for legitimate programs. For example, in the MS-DOS environment any program can dynamically invoke COMMAND.COM. Therefore any program can execute the command FOR %X IN (*.*) DEL %X to delete all files in the current subdirectory, without a user prompt. In addition, some personal computer programs bypass the operating system and directly exercise the disk controller to implement various copy-protection schemes, closely mimicking viral behavior.

A particularly pernicious example of a virus would be a compiler that is used to compile itself to produce an "improved" version, but which in fact introduces subtle bugs in all programs that are compiled subsequently, including itself. It is conceivable that the TCB itself could become infected in this manner the next time a portion of it is recompiled, since the TCSEC does not require the use of validated compilers or other program development tools even at the A1 level of trust. In particular, as Ken Thompson pointed out in his 1984 Turing Award Lecture,[10] if a compiler were modified to recognize and modify a portion of its own code the next time it was recompiled, the offending source code could then be removed leaving no trace of the modification except in the object modules of the compiler. If in addition the compiler modifications generated a security mechanism bypass or other Trojan horse function or computer virus that would be installed in specific programs (*e.g.*, all programs called "LOGIN"), the effect could be devastating. Unfortunately, producing compilers that just work as they were intended sometimes seems to be beyond the state of the art, as bugs are found in the output of most compilers, including certified Ada compilers. Detecting deliberately introduced compiler viruses would therefore appear to be a very difficult task, and one that should be addressed through the use of cleared and trusted programmers, an independently-audited formal design methodology that includes compiler source code review, independent compiler generation and object module verification,[11] meticulous configuration control throughout the life-cycle of the compiler, and a trusted application development system that is based on a high-assurance TCB that includes the principles discussed in this paper. The use of such validated compilers and other trusted development tools should be one of the first requirements for a "beyond A1" level of trust.

Although viruses must be inserted into programs that are later executed (or potentially into data files that are later interpreted, including source programs, command files, spreadsheets, and database applications) in order to take effect, the *damage* function of a virus is not necessarily confined to programs. In addition to perhaps denying service to the user by locking up the

---

9. Cohen, Fred, "Computer Viruses: Theory and Experiments," *Computers & Security*, North-Holland, Amsterdam, vol. 6, no. 1, April 1987, 22-35.

10. Thompson, Ken, "Reflections on Trusting Trust," *Communications of the ACM*, vol. 27, no 8, August 1984, pp 761-763.

11. The compiler verification team would use an independently-produced "filter" compiler or cross-compiler to compile the source of the target compiler to produce a working if unoptimized version, which would then used to compile the production version. The object modules produced by this two-pass process would then be compared to those generated and submitted for verification by the compiler development team. The compiler used to produce the working version must be kept strictly isolated from the compiler development effort itself, in order to prevent the verification compiler from introducing a virus into the production compiler. Through this technique, three independently-produced certified Ada compilers capable of compiling themselves, for example, could be used to produce and validate each other's object code as being virus-free. Note that this technique does *not* preclude a compiler with faulty or malicious source code from including a bug or even a virus in the compiled programs — it merely confirms that the object code of the compiler itself conforms to its own source code. In addition, an informal reasonableness review of the compiler-generated assembly language listing vs. the source code should be performed by the programming team, and an automatic comparison of the generated object module with the assembly language listing should be performed to detect any differences. For some additional techniques cf. McDermott, John, "A Technique for Removing an Important Class of Trojan Horses for High Order Languages," **Proceedings 11th National Computer Security Conference**, NBS/NSA, 17-20 October 1988, pp 114-118.

machine or destroying hardware (for example by setting the oscillator chip on the graphics controller board to a frequency which would burn out the fly-back transformer in a video display terminal, or by changing the disk controller parameters to values which would either make the disk unreliable or would eventually cause it to fail), both random and specific changes could be made to source program files, to spreadsheets and databases, missile launch coordinates, etc. Of all of the various ill-effects a virus or Trojan horse program could cause, a slow propagation of error throughout all of the files on the system might be the hardest to find and eradicate, and might do the most damage because it might contaminate the backup files before the effects were discovered.

Even encrypting all of the files and directories on the disk is not sufficient to prevent a random data-modification attack if the Trojan horse program can write to the disk, since the encryption will merely make the changes more random.[12]

Ideally, any new program that is acquired should be provided in source form, so that the code can be examined and recompiled to eliminate any existing viral modifications. However, even if the source code is available, misleading comments may have been accidentally or deliberately included which would mask the true purpose of the code at some point. Even with formal specifications it is very difficult to prove that a source program does exactly what it is supposed to, and even more difficult to prove that it does nothing else. At the present time, formally proving the correctness of a large assembly language program or object code module is considered beyond the state of the art, even when the programmer is actively assisting the proof process, so trying to prove something about a program without the specifications and the source code can be considered essentially impossible.

Unfortunately, the legitimate proprietary interests of the vendors prevent the source code from being available in most cases, giving rise to the possibility that even shrink-wrapped, Commercial-Off-The-Shelf (COTS) software is already infected with a virus, with the vendor being a (presumably) innocent dupe. Indeed, shrink-wrapped viruses have already struck the Macintosh and Amiga user community.[13] A new program received from an outside source should be tested as exhaustively as possible but especially if the source code is not available.

Because a virus that is inserted into a program may remain passive until some external triggering event or combination of events occurs, viruses may be impossible to detect by even the most comprehensive set of test cases. However, that should not be used as an excuse to avoid adequate testing! Although testing is not a sufficient condition to prove that a program is benign, in the absence of formal proofs of correctness it is assuredly a *necessary* condition.

The major problem in testing a suspected virus is to assure that an exact simulation of the working environment is provided during the test, and then recognizing whether any unexpected effects have occurred. For example, it is frequently recommended that the testing of potential viruses be conducted in a "sterile" environment, *e.g.*, a system which only uses floppy disks.

Unfortunately, sterile testing may *not* reveal a virus which is sensitive to the real operational environment. For example, a virus that waits until the hard disk is 75% full before destroying all of its contents, or only alters the contents of programs or data if more than 512K of memory is used, will not be detected by sterile testing. As a

12. Encrypting all of the files on the disk would prevent a virus from replacing or intelligently modifying existing programs, but only if the virus can be prevented from accessing the encryption/decryption mechanism. For example, in the case of a personal computer, if an add-on hardware board were used to encrypt and decrypt all files, and if the encryption keys never left that board and a dedicated controller were used to cause the encryption/decryption, then any attempt to exercise the disk controller directly to do reads or writes would be foiled, as only random garbage could be read or written. If in addition the special encryption/decryption software implemented a secure attribute such as READ-ONLY/WRITABLE which would require the user to enter a password before a file could be modified, then a virus or Trojan horse would be limited to attacking a file while it was being created. This type of attack would require that the operating system and/or the application programs used to create the file be modified, perhaps through some form of Terminate-and-Stay-Resident program. But if the operating system and the applications program were installed securely and write protected, the attack would become quite difficult. Of course, keeping the operating system, all application programs, and all critical data on floppy disks, protecting them with write-protect tabs immediately after updating them, and storing the floppies in a safe when they are not in use would accomplish much the same thing, but with some inconvenience.

13. Highland, Harold J., "Computer Viruses—A Post Mortem", *Computers and Security*, vol. 7 no. 2, April 1988, pp 117-121.

result, it may be necessary to offer up some "bait" to the virus, potentially allowing data destruction to occur, and then carefully checking the entire state of the system after the program terminates to see whether any files were modified or destroyed, any Terminate-and-Stay-Resident programs were installed, any changes were made to other resident programs, etc. This may require a significant amount of effort, of course, including reformatting and restoring the entire hard disk.

In addition, in order to be effective virus testing must exercise all possible parameters and data inputs, for example by systematically advancing the system date. Even then a virus that only causes damage on February 29th, or on Friday the 13th, or at 11:11:11 AM on the anniversary of some programmer's dismissal is not likely to be found.

For this reason, dynamic flow tracing techniques in combination with program disassembly should be used to ensure that every possible branch path in the program being tested has been exercised at least once. Branch paths that are never taken during routine processing should be considered suspicious and examined more thoroughly. Although the use of the CASE statement instead of convoluted IF tests is generally considered to be good programming practice, programs which make use of generalized CASE statements using indexed branch tables are potentially dangerous unless the branch index is limit tested. Otherwise, the branch table index could be modified to point to some other routine, or even some instructions embedded in data. Procedures which pass subroutine addresses as parameters are even more dangerous. Programs which dynamically invoke routines not contained within the load module should be likewise be considered suspicious, and programs which incorporate self-modifying code should be flatly prohibited. Programs whose licenses forbid such disassembly and testing techniques without offering a full warranty against consequential damages in return should be rejected as not being worth the risk.

However, even exercising a suite of test cases which causes every single instruction in a program to be executed at least once is not sufficient to prove that a program is benign, because we must consider all possible variations of data input. For example, a program for the IBM-PC might load a register from a memory address that is near the high end of memory but outside of the transient area that MS-DOS uses. In most cases the register would contain 0, since the memory location would not have been used since the system was initialized. The program would then add "N" to that variable before invoking an I/O routine to write N bytes to the disk. But if the field that normally contains 0 were changed to something else, say 30,000,000, by a cooperating virus (or inadvertently, by another program), all of the disk could be wiped out. Although loop-free programs offer some defense against such techniques, not all programs can be economically loop-free even if all programmers were disposed to be so security conscious. In addition, some basic computer instructions, notably string move operations, are inherently vulnerable to attack by changing the length operand. Unless the purpose of every single assembly-language instruction can be justified and tested with all possible variations of input, it is almost impossible to prevent such effects in a general purpose program.

Unfortunately, these techniques and others can be combined to form what might be called *binary viruses*. Like binary nerve gases, the individual components of the binary virus are entirely benign, but when both are run together, or one after the other, the results are fatal. One program might contain the viral reproduction mechanism and another program could contain the signal which would activate it. A third program might trigger the actual damage function. If data-sensitive rather than instruction-flow sensitive programming techniques were used, the individual programs could be tested forever, even using branch-tracing techniques, yet the ill-effects would never be revealed until at least two programs were combined.

Detailed, instruction-by-instruction simulation testing may be helpful in revealing such data-sensitive programming techniques, and of course these techniques are often used in debugging complex programs. In particular, if the simulation includes traps to detect various out-of-range conditions, such as reading locations that have not been set, or writing outside of the allocated memory for that program, these conditions may be detected. Even then, a clever virus could perhaps detect the fact that it was being simulated by checking the clock or reading

a buffer after a read had been requested but before the hardware would normally have had time to respond. Realtime hardware emulation techniques, logic analyzers, bus monitors, etc., can be used, but at an increasingly greater cost in time and hardware.

We are therefore forced to conclude that although viral filter programs may be devised that will detect specific viruses or classes of viruses, it is almost impossible to discover or stop all such attacks through a practical amount of testing or inspection.

Instead, we should build memory segmentation techniques and read/write protection into the basic hardware, and construct the operating system so that it limits the amount of damage that a virus can cause. In addition, it is obviously desirable to limit the initial introduction of viruses into the system by only admitting programs from presumably trustworthy sources.

It must be emphasized that the viruses and Trojan horse programs that have appeared to date have been relatively unsophisticated in terms of their construction and effect. In particular, the damage done by the attack has usually been severe and obvious. More subtle attacks can probably be anticipated, both by amateurs and professionals, with both the security and integrity of data being more directly threatened, but less visibly so.

Because Trojan horse programs and computer viruses cannot be reliably detected in advance of their actions, it is vitally important to use a defensive *containment mechanism* that can prevent or at least detect any change to a subject or object, but which cannot itself be subverted, in order to prevent such programs from affecting the rest of the system.

3.3 Integrity Containment Mechanisms

In order to address the requirement to defend against such attacks, Clark and Wilson introduced the concept of a Constrained Data Item (CDI) together with a trusted program called a Transformation Procedure (TP), and their rule E2 states that "The system must maintain a list of relations associating triples of the form: (UserID, TP_i, [CDI_a, CDI_b, CDI_c, ...]), which relates a user, a TP, and the data objects that TP may reference on behalf of that user. It must ensure that only executions described in one of the relations are performed."

However, although a list of such triples can easily be used to grant *permission* for a given TP under the control of a given user to access a given set of CDIs, a much more general **containment mechanism** (similar to the kernel of the TCB that enforces the mandatory access controls) is required in order to be able to *prove the negative proposition* that no other program or agent, either internal or external, could possibly access or alter a CDI, *except* as permitted by the list of access permission triples. The difference is crucial, for while access *permission* can be granted (by individual authorized users) through Discretionary Access Controls that could be supported by a Class C2 TCB in TCSEC terminology, access *denial* must be enforced by a system of Mandatory Access Controls which requires the functionality and increased assurance of a Class B1 or higher TCB, and which can only be overridden by the security officer or similarly privileged users.[14]

In addition to the general difficulty of proving a negative conjecture, implementing a system of mandatory integrity controls based on the use of Access Control Lists alone would be impractical if there are a large number of hosts connected by some kind of a network, because of the management problems involved coordinating all of the triples in the system and keeping them up to date. In particular, it would be very difficult to implement a system that would control the creation and dissemination of *new* objects throughout the global network. **In the case of a network, all of the Access Control Lists throughout the network would have to be correct in order to prevent an unauthorized modification.** The synchronization and management of the required databases throughout the network would be a very difficult

14. The requirement for a B1 or higher TCB does not necessarily mean that a conventional label-based system must be used. A capabilities-based operating system such as the KeyKOS system being developed by Key Logic and targeted for the B3 level of assurance would provide the strong firewalls necessary to satisfy this requirement as far as the local TCB is concerned, but the difficulty of securely coordinating the various capabilities across a global network would remain.

problem, since the Access Control Lists themselves must be Constrained Data Items.

Further, Access Control Lists can neither prevent nor even detect the accidental modification of data or programs by hardware or software error, nor the deliberate modification of such objects during the transmission or storage of the objects outside of the TCB. In the general case of TCBs which are connected via untrusted networks, or networks which include both trusted and untrusted hosts, it would be almost impossible to prevent the unauthorized modification of a data object while it is being transmitted or stored outside of the security perimeter of a trusted host. For example, information that is stored on a floppy disk could presumably be modified by anyone with access to a personal computer.

Finally, the use of Access Control Lists without other more powerful integrity containment mechanisms would require that many sections of the operating system, including such general purpose utilities such as Move/Copy, be trusted for all users and for all files. As a result, we are either faced with the daunting task of proving that all such programs do only what they are supposed to do and nothing else, or else accepting the risk that they contain a virus or Trojan horse program.

In summary, although the Clark-Wilson database of triples is a powerful notational device and may also be appropriate for certain database implementations, it is not readily adaptable to a network implementation. **An approach to integrity that cannot cope with floppy disks or untrusted networks is impractical.**

In a sense, this problem is similar to the problem of writing information onto an unreliable medium. We recognize that errors may occur during the writing or reading steps, and we use checksums to limit the occurrence of undetected errors. Although random data errors and hardware failures can be difficult to prevent, in most cases they can be detected by the use of simple checksums such as the CCITT CRC-16 cyclical redundancy check polynomial. In many cases they can not only be detected but corrected automatically through the use of error detecting and correcting codes (Forward Error Correction), or through protocols which require that the message which is received in error be retransmitted by the sender.

Deliberate attacks by either external or internal agents are much more difficult to prevent or even detect, however, because it has to be assumed that the perpetrator knows the checksum algorithm and/or error-correcting code and can manipulate the information so that the erroneous data appears to be correct. Only a strong cryptographic checksum such as a Manipulation Detection Code (MDC) (which is protected by an external encryption scheme such as a digital signature) or a Message Authentication Code (MAC) (which uses a secret cryptographic key to authenticate both the sender of the message and the message contents to the receiver) can detect such an attack.

Therefore, instead of attempting the almost certainly futile task of preventing such a change throughout the entire network (including the receiving TCB) by controlling who can write or alter a Constrained Data Item, it might be better to rely on **detection mechanisms** which would invalidate the data if such a change were to occur, plus the use of normal recovery/restoral mechanisms to repair the damage. Instead of relying on global coordination in order to attempt to control which individuals and processes can access a data item, we could split the control responsibility into two parts:

1. Conventional security mechanisms, including encryption, can be used to prevent unauthorized individuals from reading or executing a data object. The determination of what users will be able to read or execute the object can be made by the *originator*, and enforced by cryptographic means that include end-user to end-user file encryption.

2. Cryptographic checksum and digital signature mechanisms can be used to allow the *recipient* of the information to determine its believability, including an assessment of the credibility and/or authority of the originator and the absence of unauthorized modification.

To reiterate this basic principle: **The *originator* of an object should be responsible for assuring its security. The *recipient* should be responsible for determining its integrity.**

A cryptographic checksum checker that is contained within the protected TCB can be used to protect against an undetected modification of a data item, even if the modifications were to occur while the object was being transmitted, transported, or stored outside the security perimeter of the TCB. By securely affixing a checksum to the object when it is first created (within the TCB), and then recomputing the checksum over the data item and comparing it to the recorded original whenever the object is brought into the TCB again, any modifications can be quickly and accurately detected. The use of a digital signature mechanism of the type proposed by Rivest, Shamir, and Adleman,[15] the so-called RSA algorithm, can also provide non-repudiation of the authorship and contents of the particular data item, if that is required.

Although the checksum and digital signature technique could be applied at any level of abstraction from a field within a record, to individual records, individual files, collections of files (a volume), etc., in most cases protection at the file level will give adequate granularity without an excessive amount of overhead.[16] If the checksum is calculated at the file level, the checksum calculation must be chained from record to record within the file in such a manner as to prevent entire records from being added, deleted, modified, repeated, or rearranged.

If all of the files accessed by a program were protected by a checksum and digital signature recorded in a file label by the TCB and checked each time a file was accessed, a very significant degree of protection against undetected modification would be provided. The extension to cover multiple TCBs and data that is stored or transmitted over non-secure media would then be straightforward, whereas the coordination of Access Control Lists and similar mechanisms to prevent such modification across multiple TCBs would be nearly impossible.

Although it could be argued that an integrity label that contains a cryptographic checksum or digital signature is simply a different form of Access Control List, there is an essential difference. In the case of a large network, all of the Access Control Lists throughout the network would have to be correct in order to *prevent* an unauthorized modification. With a digital signature mechanism, however, once the originator signs the object any recipient of the object can verify its contents at any time, and thereby *detect* any unauthorized modification.

In addition to protecting data while outside the security perimeter of the TCB, a digital signature and checksum technique may facilitate the design and certification of the TCB itself. A cryptographic checksum not only provides an indisputable indication if any tampering has occurred, it also provides a quantifiable level of confidence that tampering has not occurred.[17] It may therefore be significantly easier to evaluate the integrity of a TCB which incorporates a cryptographic checksum and digital signature technique as part of its Mandatory Access Control and Trusted Distribution procedures than to attempt to prove the negative conjecture about the entire system, *i.e.,* that no modification could have occurred to the TCB or any associated application programs or data, nor to the various security labels associated with an object.

However, it is obvious that the use of a digital signature by itself is not sufficient to guarantee the integrity of an object, any more than a signature on a check necessarily means that the signer is authorized to withdraw funds from that account, or that there is money in the bank to cover it. We still have to ensure that the

15. Rivest, R. L., A. Shamir, and L. Adleman, "A method of obtaining digital signatures and public key cryptosystem," *Communications of the ACM*, vol. 21, no. 2, Feb. 1978, pp 120-126.

16. The strength of the RSA digital signature algorithm depends on the difficulty of factoring a large composite number. Present-day techniques allow the economical factoring of 100 decimal digit numbers, or approximately 330 bits. The difficulty of factoring roughly doubles for every three additional decimal digits in the composite number, whereas the expense of factoring a given number has decreased by roughly an order of magnitude per decade. Maintaining adequate security and integrity for stored records therefore requires many hundreds or even thousands of bits in each digital signature, if the secrecy and/or integrity of the object is to withstand four or five decades (for a significant fraction of an individual's lifetime) of improvement in factoring technology. The design implications of this amount of overhead are considerable.

17. A cryptographically strong 128-bit checksum would require generating, sorting, and comparing approximately 2^{65} or 10^{19} variations of two different sets of data objects in order to have approximately a 50% chance of spoofing the system via a brute-force or so-called Birthday Attack. Although the generation of that many variations might be marginally feasible, storing, sorting, and comparing that many checksums is beyond the present state of the art.

originator of the object was appropriately authorized and that the inputs to the creating process were valid as well as the process itself, and we will discuss those points in detail later on.

Giving the recipient of a data item the primary responsibility for determining whether an object has been tampered with (by verifying the checksum) and whether it was created by an authorized individual (by validating the digital signature) has another significant advantage — it allows us (the system) to change "our" minds, and to reject data items that might have previously been viewed as acceptable. If necessary, erroneous or malicious programs, incorrect or modified data items, and untrustworthy data originators could be "blacklisted" so that they would no longer be honored. Interestingly, viral-type dissemination mechanisms could be used to "spread the word" throughout a network, using an authenticated blacklist message that is digitally-signed by an appropriate authority.

Any new object that is entered into the system from the outside should therefore be "sponsored" by an authorized user who takes some responsibility for introducing the item. If the object is later determined to be a Trojan horse program or otherwise corrupt, there will at least be some clear evidence as to where it came from, and a basis for administrative action or even prosecution.

The fundamental requirement for system and application integrity in a global, untrusted networking environment, *i.e.*, to ensure that an object was created by an authorized user or process and has not been modified since, can now be summarized by two rules:

Rule 1 — Immutability:

> It shall be possible to verify with a very high degree of assurance, *e.g.*, through the use of a strong cryptographic checksum, that the contents of an object (data or program) have not been changed, either accidentally or deliberately, since the object was created, brought into the TCB, or last modified by an authorized user. Objects whose checksums indicate tampering has occurred shall be rejected.

Rule 2 — Attribution:

> All programs and data introduced into the TCB or subsequently changed must be explicitly attributed to an authorized user or "originator."[18] Objects which cannot be attributed to an authorized user shall be rejected.

3.4 A Definition of Integrity

A number of different definitions of integrity have been put forth by different authors, and there has been relatively little agreement as to what is meant by the term. Some definitions (notably those before Clark-Wilson) tend to define integrity in terms of the absence of modification (immutability), with or without the notion of attribution. Other definitions tend to highlight the notion of *duly-authorized* modifications, and emphasize the "correctness" of the information with respect to some external "reality." In a sense, the argument over terminology may reflect a disagreement over epistemology and the nature of truth which appeals to different dictionaries cannot resolve. While it doesn't really matter whether we call this "thing" that we have been talking about "integrity," or "verity" (which perhaps comes closest to the notion of correspondence with an external reality, although it is an uncommon word), or even "hozzanga," a concrete definition should at least be provided for the sake of clarity:

Integrity is a property of those *processes* (subjects) and *information* (objects) that meet an *a priori* expectation (specification) of a level of quality that is considered acceptable for a particular application.[19]

A number of observations can be made regarding this definition:

---

18. This rule also addresses the Clark-Wilson requirement for an Integrity Verification Procedure, which is intended to ensure the integrity of data that is input to the system by validating it against the reality of the external world, *e.g.*, by requiring that the inventory records accurately reflect what is on the shelves.

19. Adapted from a definition suggested by Dr. Willis Ware, in Robert Courtney, Jr.'s "Some Informal Comments About Integrity and the Integrity Workshop," distributed to participants at the NIST Invitational Workshop on Data Integrity.

1. Integrity is not quality *per se* (i.e., accuracy, precision, timeliness, completeness, consistency, robustness, pedigree, etc.), but rather **the extent to which those qualities taken together is considered adequate for a given purpose.**

2. The definition of acceptable quality must consist of **a set of syntactic and semantic rules** that can be either static or dynamic, but are specific to a given application or purpose.

3. The integrity of *information* (objects) may be intrinsic to the data (within a given context), but is more often extrinsic, *i.e.,* a function of the input data and the subject responsible for the creation of the object.

4. The more tolerant the syntactic and semantic rules are for a given application, the more likely it is that they will be satisfied by a given program or set of data. As a result, the integrity of the information may be high, but the quality may be low.

5. The integrity of *information* means more than just checksums on data. The integrity of information may, but does not necessarily, change as the information changes form. For example, if a message is received in 7-bit ASCII code, that message can be translated with absolute accuracy into 8-bit ASCII, and except for a few graphic characters that do not have an equivalent mapping, it can be translated into 8-bit EBCDIC. But a message that uses both upper and lower case characters cannot be accurately translated into Baudot or Morse code, for the EMPHASIS given by capital letters will be missing if only capital letters can be represented. If a process changes the form of information, we must somehow address the integrity of that process to ensure that the meaning of the information is not changed, or changed in a measured way.

6. Integrity is not reliability. Reliability applies to hardware processes which may fail due to random, non-deterministic causes. Software is perfectly reliable and deterministic, in that it never fails or deteriorates by itself — a program will always behave in exactly the same way if it receives exactly the same inputs.

A metric for integrity must consist of (1) a specified set of rules and (2) a measure of trust or confidence, (3) that is determined by an acceptable authority.

3.5 Integrity Domains

Clark and Wilson did not define integrity, but the major thrust of their paper was that it is necessary to ensure that certain objects (data or programs) or modifications to previously existing objects are the result of actions by authorized users, effected through sufficiently trusted programs as to guarantee the integrity of the result. They proposed to accomplish this by requiring that so-called Constrained Data Items (CDIs) can only be manipulated by a restricted set of Transformation Procedures (TPs), in accordance with rules which ensure that all such transformations are *"well-formed"*.

Clark and Wilson did not define what constitutes a "well-formed" transaction either, presumably leaving that to the human certifier of the TP. However, their concept is obviously one that involves a completed state transition, and seems to have been highly influenced by their model of double-entry bookkeeping as the archetype of a high-integrity process. Although they do not stress the point, in some systems which operate in a distributed environment it may be particularly important that the well-formed transaction be carried out as a single atomic activity, so that the two coordinating processes know exactly what the state of the system is. This frequently takes place through a two-step procedure where the process A essentially says "Prepare to commit X," process B replies "Committed to X," and then Process A instructs either "Commit" or "Abort," and B acknowledges. The Clark-Wilson rule C2 therefore requires that the TP must take a valid Constrained Data Item (CDI) from one valid state to another, while their rule C5 states that a TP must take an Unconstrained Data Item (UDI) (*e.g.,* untrusted user input) to a valid state (a CDI), or else perform no transformation (a null or error output, presumably).

In combination with their rule C1, which requires that a set of Integrity Verification Procedures (IVPs) must properly ensure that all CDIs are in a valid state at the time the IVP is run, they suggest that a process similar to mathematical induction will result: If the system can be shown to be in a valid state initially; if all UDIs are edited or transformed into CDIs by a TP; and if all TPs are certified to be valid (*i.e*,, they must take a CDI to a valid state, given a valid initial state); then the system will remain in a valid state after the TP has executed. Indeed, this "proof by induction" is seemingly one of the most attractive results of their paper.

Unfortunately, the Clark and Wilson inductive model is not very useful, for it cannot be assumed that a TP has knowledge of the entire system. To do so would violate virtually all modern precepts of modular system design, controlled interfaces with information hiding, etc., as well as requiring impracticably large TPs. Instead, it must be assumed that a TP can only have knowledge of its explicit data inputs (either CDIs or UDIs) plus its built-in (programmed) syntactic and semantic processing rules.

Since a TP cannot have a global view of what constitutes a "valid" data input but can only process data in accordance with the information it has available, it follows that the "valid" output of one TP may be judged invalid by another TP that has a different set of inputs and/or syntactic or semantic rules.

An example of such an apparent contradiction would be a Pascal source program that erroneously called the external ARCSIN function by its Fortran name ASIN, instead of the Microsoft Pascal library function ASSRQQ. The compiler would consider the reference to ASIN to be perfectly valid, since it was defined with an EXTERN reference, but the linkage editor would not be able to find ASIN in the Pascal subroutine library. The output from the compiler is syntactically and semantically valid as far as the compiler (or even the programmer) knows, but the overall state of the system will not be valid unless the CDI output by the compiler is rejected by the link editor.

It is therefore necessary to combine Clark and Wilson's rules C2 and C5, and require that a TP perform a valid transformation, or else no transformation, given either a CDI or a UDI. The distinction between a CDI, which has already passed at least some validity tests, and a UDI, which may not yet have passed any screening tests, is therefore only a matter of degree with respect to the internal functioning of the system. The alternative would be to perform some sort of an information flow model verification of the entire system, recognizing that some CDIs are more valid and/or more critical than other CDIs, at least with respect to the overall system.

Thus it can be seen that "validity," like "beauty," is not absolute but only relative — the question always has to be asked, "Valid with respect to what criteria?" As a result, it is very important to carefully describe the set of allowable inputs and the syntactic and semantic rules that have been used to establish the relative validity of a CDI, or more pragmatically, to establish what validation program(s) and what data were used to produce a given CDI.

We will define the set of allowable inputs to a process together with the syntactic and semantic rules used to validate or reject those inputs and optionally produce one or more outputs an *integrity domain*.

The Clark-Wilson induction must therefore be modified to state that if all the IVPs and TPs are certified as valid *with respect to a particular domain*, and if all the CDIs are valid with respect to that same domain, and if only TPs that have been certified valid are allowed to access the CDIs, then all of the CDIs will remain valid with respect to that domain (only).[20]

Because it can be shown that there are programs whose properties are theoretically indeterminate (*i.e.,* we cannot even decide in advance whether the program will terminate or not, much less what the final state of the system will be if it does), it is often not possible to say with certainty whether a given program will obey the syntactic and semantic rules or not. And as both a theoretical and a practical matter it usually not possible to

20. Even then we must be careful to state that the TPs must be certified with respect to their *dynamic* properties. Either the TCB or the TPs themselves must enforce the proper sequencing of processes against data and protect against processes being run too few or too many times, or in the wrong order, or the results may still be invalid.

test a program exhaustively, either. The best we can do in general is to estimate the *probability* that the rules will be obeyed, either by pragmatically observing the behavior of the program over a wide variety of inputs, or else by applying human experience and judgment and any available formal design and testing tools in order to arrive at an educated guess.

We can therefore conclude that the integrity of a process is probabilistic, *not* binary, and can only be defined relative to a specified integrity domain. Further, although the process that produced a given data item either did or did not conform to the syntactic and semantic rules, and therefore it could be argued that the data item either has complete integrity or it doesn't, we have no operationally useful way of determining that fact, either *a priori* or *a posteriori*. For that reason we should consider the integrity of an object to be probabilistic as well.

When we proposed a definition of integrity, one of the aspects of quality that was mentioned was timeliness. Although Clark and Wilson discuss the necessity of running an IVP periodically, their model does not specifically address the possibility of information becoming stale. We will assume that the rules that make up an integrity domain are invariant over time, although they may be dynamic in that they involve previous data inputs, may maintain a history, and may include time as a parameter. On the other hand, although we may have a very high confidence that the computer model was calculated correctly originally, we may have a steadily diminishing confidence that the model continues to correctly mirrors reality if the reality can change independently of the model.

We can solve this problem by making the confidence or probability portion of the definition of integrity a time-dependent parameter, so as to force the correspondence between the model and reality to be periodically revalidated by rerunning an IVP.

Finally, as we concluded in our discussion regarding viruses, we must provide an integrity confinement mechanism that will ensure that strong "firewalls," including such techniques as memory segmentation and read/write protection, are provided to prevent objects from being tampered with and subjects from being misled while they are contained within the TCB.

For these reasons the Clark and Wilson rules C1, C2, and C5 will be replaced with the following:

Rule 3 — Validation:

Transformation Procedures (TPs) are subjects (processes) that are trusted to process objects (information) within a defined scope, in accordance with the rules of a specified integrity domain. TPs shall either validate or reject objects that fall within the scope of their integrity domain, or else rely upon the TCB to provide appropriate integrity constraints so that the TP can only access objects which have been previously validated by other TPs and not modified since. No TP shall be allowed to create, modify, rename, or delete an object whose integrity is outside the scope of the TP's certification. The integrity of an object may diminish over time if the representation provided by the computer model begins to diverge from the external reality, or vice versa.

Rule 4 — Certification:

Transformation Procedures (TPs) shall be certified by an approval authority to perform their function correctly (within some specified probability of error), properly screening data inputs and producing the prescribed outputs in accordance with a well-defined set of syntactic and semantic rules that constitute the integrity domain of that process.

There is a considerable body of art and science that has been developed concerning the problem of how to specify, design, develop, test, and certify application programs to be "correct." Although much improvement could presumably still be made in this area, many books have been written about the subject and we will not concern ourselves further with this important and admittedly difficult task. Instead, we will continue to focus on the confinement mechanisms that a Trusted Computing Base must present in order to assure that certified programs will be executed correctly, *i.e.*, in accordance with

the constraints of Rule 3 and other principles yet to be discussed.

3.6 Partition of Change

In addition to constraining what data a TP is allowed to process, Clark and Wilson state that the system must be capable of enforcing a principle of separation of duty between duly authorized users, such that certain programs can only be executed by certain individuals and collusion is required to circumvent these controls.[21] Although collusion is certainly not impossible, observation of human nature over many centuries has shown that it is much more difficult to get two people to agree to commit an illegal or unauthorized act than one person. Because conspiracy to commit a fraud is generally a crime even if the act is not carried out, the person proposing the collusion can never be sure that the other person will not turn him in for the reward, rather than acting as his partner in crime. It is best, of course, if the separation of duty involves two people from significantly different backgrounds and responsibilities, so that they are less likely to succumb to a common temptation.[22]

Lee[23] has pointed out an interesting difference in the assumptions made about the trustworthiness of individuals between the military and the commercial security environment. In the military it is assumed (more or less) that users are trustworthy to the extent of their clearances, and Discretionary Access Controls and compartmented controls are primarily used in the exception to limit the dissemination of extremely sensitive information to those who don't have a need-to-know. In the commercial world, on the other hand, the routine imposition of the separation of duty requirement is a reflection of the perhaps cynical assumption that eventually every man has his price, and that at some level all users must be considered potentially untrustworthy.

However, there is more to the separation of duty requirement than just the need to impose checks and balances and ensure that conspiracy is required to circumvent the controls, as described by Clark and Wilson. In this respect we must respectfully disagree with Johnson,[24] who claims that "Background investigations for criticality are unnecessary. A person with a Top Secret clearance can be trusted to deal with Highly Critical data and processes. A person with a Secret clearance can be trusted to deal with critical data and processes." Unfortunately, someone may be loyal to his country and yet be an outright thief, a psychopath, or an incompetent fool.

For that reason one of the fundamental issues with respect to separation of duty and the concept of duly authorized users has to be the question of **our confidence in the competence and judgment of the individual performing in the assigned role**. A person may have a Top Secret clearance with all sorts of special "tickets," but that doesn't mean that he is necessarily qualified to operate a nuclear reactor, to perform open-heart surgery, or even to have a driver's license. For example, nuclear weapons crews not only have to be technically qualified to perform their assigned functions, but even more important for the safety of the rest of us, they undergo frequent psychological screening to ensure that they are both sufficiently stable and well-motivated that they will not release those weapons except under very carefully defined and controlled conditions.

An integral part of the concept of Separation of Duty is the notion of *least privilege*. The TCSEC requires the implementation of this concept in Class B3 TCBs when it requires that "the

---

21. This requirement may be less important than in the past, once digital signatures and other non-repudiation measures are available to provide irrefutable evidence of who authorized or caused a given action. But an ounce of prevention is worth a pound of cure, so separation of duty is expected to remain an important tool for the prevention of fraud and usurpation of authority. After all, if a company fails because of embezzlement, or nuclear missiles are launched without proper authority, the criminal conviction of the miscreant would provide cold comfort to the survivors.

22. Resorting to caricature to make the point, instead of having two Captains who both went to the Air Force Academy and who report to the same commander serve as nuclear missile control officers, it would be better to have Gen. Curt Lemay and Jane Fonda sharing joint control.

23. Lee, T. M. P., "Using Mandatory Integrity To Enforce "Commercial" Security," **Proceedings 1988 IEEE Symposium on Security and Privacy**, Oakland, CA, 18-21 April 1988, p. 140-146.

24. Johnson, Howard L., "Security Protection Based on Mission Criticality", **Fourth Aerospace Computer Security Applications Conference**, Orlando, FL, December 12-16, 1988, Computer Society Press of the IEEE, Washington, DC, 1988, pp 228-232.

functions performed in the role of a security administrator shall be identified. The ADP system administrative personnel shall only be able to perform security administrator functions after taking a distinct auditable action to assume the security administrator role on the ADP system. Non-security functions that can be performed in the security administration role shall be limited strictly to those essential to performing the security role effectively." These requirements are no less important for many "commercial" systems.

We therefore have the following requirement:

Rule 5 — Separation of Duty:

> It shall be possible to constrain the approval authority for a trusted process from executing that process, and in general to specify which authorized users and/or combination of users (either as a class or individually) may or may not initiate the execution of certain processes and/or create or modify certain data. In addition to the requirement to confine programs to reading and writing only certain specified data items, it may be necessary for the TP to know which human user initiated the process, as well as which human user(s) created or approved the data object(s) input to the process, and to be able to limit the acceptability of such data to specified individuals or combinations of individuals, or to exclude data created by specified individuals.

It should be pointed out that the requirement for separation of duty imposes a subtle requirement on the system with respect to the global identification of users. According to the TCSEC, unique identification of authorized users is required for all TCBs of class C2 or higher. However, only at the B3/A1 level of trust is the requirement imposed to be able to *exclude* a given individual or group of individuals from accessing an object (discretionary access control). As a result, at the B2 level and below, naming techniques may be used that involve the membership of a person in some group or which qualify the name with a host name or network address, in order to ensure the uniqueness of the name. Although such an approach may properly qualify the user from the standpoint of *inclusion*, it does nothing to rule out the possibility that the same user may belong to a different group, or have an alternative network address through which he or she could access the TCB. In this manner a single user could appear to be two different individuals and thereby avoid the separation-of-duty exclusion.

3.7 Identification and Authorization

The only way to solve this access exclusion problem is to use a global identifier that is guaranteed to be unique over a specified domain, *e.g.*, all of the TCBs under the aegis of a single Designated Approving Authority (DAA) or multiple DAAs cooperating under a Memorandum of Agreement. In the civilian world, the DAA would correspond to a corporate-level security officer or system administrator charged with overseeing the installation and operation of a network or system of TCBs. Independent corporations and/or Government agencies could also agree to honor each other's security controls via a Memorandum of Agreement, for example for the duration and scope of a teaming agreement or subcontract. Unfortunately, the practical and political problems associated with creating and coordinating a master database to ensure that all UserIDs are globally unique are probably insurmountable.

In particular, the use of the U.S. Social Security Number as a global identifier is not acceptable, because it is not guaranteed to be unique (two people may have the same number), because it is too easy for an individual to obtain more than one such number, and because it isn't universal (a problem both for multinational corporations and for the military, which has to interoperate with NATO and other supranational forces.)

Instead, if users were required to register with their own DAA-accredited trusted registrar and produce adequate physical identification documents including their birth certificate; then their full name at birth, their mother's full name, and the date, time, and place of their birth could be recorded in a uniform, blank-compressed format and the totality of that information could reasonably be assumed to be globally unique. Multiple registration would thereby be prevented, assuming that corroborating evidence (fingerprints, footprints, grade school pictures,

the testimony of friends and neighbors, etc.) are presented as required to confirm that the user was presenting his or her own birth certificate. In addition, this technique would continue to assure the uniqueness of the individual despite marriage, divorce, name changes, etc.

Because such a lengthy string of data would be inconvenient in use, and also because its widespread use might violate various privacy laws, it would be desirable to transform such a string into a unique identifier that does not in itself compromise the individual's privacy. A 128-bit (16-byte) ID string could be produced by applying a one-way cryptographic transformation or checksum technique to this information. This ID string or "cryptonym" could then be used in place of or in addition to the person's full name to assure global uniqueness. The number of people on earth would have to increase to the square root of 2^{128}, or approximately 10^{19}, before there would be a reasonable chance of even two people having the same identifier.[25]

Although the use of a system of trusted registrars could assure the uniqueness of an individual's identity within the domain of cooperating DAAs, it will probably not be feasible to have those registrars be responsible for authenticating all of the particular rights, privileges, and authority conferred upon that individual, since those privileges may be application-dependent and difficult to coordinate globally. Instead, assuming that an individual may be mobile with respect to the TCBs deployed throughout a network, it would be desirable to provide an individual's clearance, rights, privileges, and authority in the form of one or more machine-readable and/or human-readable *certificates* which the user can carry with him on something like a credit card or floppy disk, which would be unforgeably authenticated (digitally signed) by the appropriate authorities. Presumably an authorized sponsor such as the user's security officer would vouch for the identity (and perhaps the security clearance) of the individual, and assuming that the security officer's own credentials were confirmed satisfactorily, the user's identity would be confirmed by a digital signature process, based on the user's physical possession of his certificate (something that he holds) plus the knowledge of a secret passphrase or Personal Identification Number (something that he knows). As biometric identification devices such as fingerprint readers become more economical, the biometric identification information could be included in the certificate and digitally signed as well.

Once the identity of the individual has been established and the authority of the various individuals in the chain of command has been confirmed, it would be possible for a centralized authority (such as a Key Management Center or KMC) to summarize and confirm the rights and privileges of that individual in one digitally-signed certificate. This top-down approach to authentication would have the advantage of speed of confirmation since only one digital signature would have to be checked.

Unfortunately, the top-down approach would mean that the details of what physical evidence of an individual's identity was examined, who authorized an individual to do what, and to what degree the authorizers can themselves be trusted; would all be hidden from view from the user who is trying to decide whether to believe another user. Only the ultimate yes/no decision made by some bored clerk would be available to the user who questions another user's *bona fides*. While such a scheme might be suitable to protect national-level SECRET and perhaps even TOP SECRET data, it would clearly not be adequate to address the delicate question of whether someone working for the CIA was in fact a "mole" planted by a foreign intelligence service.

A better alternative would therefore be a scheme for bottom-up authentication that is based on the use of a public-key or digital signature algorithm. The user would locally (and securely) generate a pair of public/private encryption/decryption keys and a pair of public/private authentication/

25. There are still some societal issues that would remain with this technique, such as the desire of many individuals for anonymity and/or privacy, and the extent to which the use of *any* globally-unique identifier would infringe upon those "rights." On the other hand, the right of society to assume that an individual will be responsible for his actions must be considered, and the right of a person not to have his good name impugned by someone who has the same or similar name is an equally valid concern. It is often necessary for those with common names to file affidavits swearing that they are not the "John Jones" against whom various liens or judgments have been filed. Finally, there are some instances where false identities or pseudonyms are officially sanctioned. These include sealed adoption records, the federal Witness Protection Program, undercover police officers, and various activities within the intelligence community. It is not yet entirely clear how to handle these problems, but presumably the authority which issues the pseudonym would bear some responsibility for its misuse.

verification keys. A local authority or registrar would then attest to that user's identity and perhaps his privileges, together with the possession of the public encryption key and public authentication key by that individual. If necessary, a precise summary of all of the physical evidence examined could be included in English in the authorization text. The registrar's own authority and digital signature would be authenticated in turn by an appropriate authority, up the chain of command to the DAA or his surrogate, *e.g.,* the system administrator. The public authentication key necessary to validate the digital signature of the DAA would be securely installed in all of the TCBs under his jurisdiction, just as in the top-down case.

The difference between the top-down and bottom-up approach is that all of the intervening authorizations and digital signatures would be included in the chain of authorization in the bottom-up case, and the decision as to the acceptability of the evidence would rest with the user who wished to communicate with or believe that user. Once the accepting user made the decision as to whether to accept a user's *bona fides*, he could authorize that correspondent himself, using his own private key, and thereby be able to revalidate the user in the future with only one step.

In addition to relegating the decision process to the user, the bottom-up approach has the advantage that multiple independent chains of command may be accommodated, for example where two peer entities agree to honor each other's affidavits through a Memorandum of Understanding. A practical example would be the case where two corporations agree to form a team or undertake a contractor-subcontractor relationship.

Finally, and this may be the most important practical point, the bottom-up approach does not require the creation and administrative expense of a centralized Key Management Center, *which must be trusted by all parties who wish to communicate securely.*

3.8 Roles and Aliases

Although it is essential that an individual's unique identity be used in deciding questions of discretionary access control and discretionary integrity control, as well as in auditing that individual's actions, from time to time the individual may legitimately take on various **roles** and **aliases** in addition to his own unique identity.

The concept of an alias is closely allied with the concept of a group. In situations where there is a certain interchangeability of personnel, any member of a group may be allowed to access specified objects, even though such objects are normally encrypted using the public key of a specific individual. Messages addressed to a functional title such as Manager, Department XYZ, for example, may be read by anyone authorized to read such messages. Rather than the all too common practice of sharing passwords (a practice which should eventually be made impossible through the use of biometric identification devices), a manager may explicitly delegate the opening of departmental mail to a secretary or administrative assistant, with instructions to route it to the appropriate individual for disposition. The secretary or administrative assistant would then log on with permission to access any of the manager's incoming correspondence under the alias of the manager, but messages marked "Personal" could only be read by the manager himself.

Although the use of an alias may authorize someone to *read* someone else's messages, the person acting as an alias would not be allowed to *sign* the principal's outgoing correspondence, unless that person has explicitly been granted permission to act as an agent for or in the **role** of the principal, typically under a Power of Attorney or Delegation of Authority instrument. Again referring to functional titles, it is possible that an individual is assigned a particular position or role for only a limited time, ranging from a particular shift in the case of a Duty Officer to an indefinite period in the case of a delegation of authority.

Finally, in order to limit an individual's responsibility and authority (and likewise his liability), certain caveats may be associated with an individual's digital signature, including a range of validity dates. An individual might want to limit the liability associated with the use of his digital signature for personal expenses to a few thousand dollars, while in his role of a Corporate Officer he might sign checks and other financial instruments whose value might be in the millions of dollars.

For these reasons, it is necessary to clearly establish what role a given user is executing when performing certain actions, and his or her authority to perform in that role must be confirmed by an explicit document of authorization. The user will normally be required to sign (either digitally or on paper) an "Affidavit of Legal Mark," which promulgates the user's public encryption and public authentication keys and provides legal notice that the user will be accept responsibility for his digital signature as though it were his written signature, subject to certain caveats with respect to that role. Such a document should be independently witnessed and notarized in order to confirm its validity, and if specific authority is claimed the grantor of that authority must sign the Affidavit or another authorizing instrument and must himself be authorized, ultimately by the DAA.

In addition, the use of a role or alias must not defeat the intent of the Separation of Duty requirement. That is, an individual cannot circumvent a requirement for two unique signatures by signing once as John Doe, and again as Manager of Department XYZ. Similarly, if John Doe is excluded from some action (perhaps because as Manager of Department XYZ he was the certifying authority for some program, and therefore is not permitted to execute that program), then he must also be excluded in his role as Manager of Department XYZ. However, a previous or subsequent Manager of Department XYZ would not necessarily be excluded.

In other words, although both roles and aliases could be thought of as defining a *virtual user*, the separation of duty requirement applies to *real* users.

We therefore have the following rules:

Rule 6 — Identification:
All users of the system shall be enrolled into the system and uniquely and unforgeably identified by a trusted registrar, so that their actions may be controlled and audited by an independent auditor. The use of an alias or role by a user shall not exempt that user from any Separation of Duty requirements.

Rule 7 — Authorization: An individual's clearances, rights, privileges, and authority shall be authenticated as having been granted by a duly authorized individual. The identity and authority of the individual granting such authorization, including the registrar who enrolls an individual in the system, shall in turn be confirmed by his or her superior in a chain of command. Any caveats associated with the individual's digital signature should be carefully noted before accepting the validity of that digital signature.

3.9 Sequencing and Program Confinement

Clark and Wilson's rules do not specifically provide a mechanism to enforce the proper sequencing of transactions, nor to confine the processing of data to a particular time of day or day of the week, but it is clear that programs must be applied to data items (or vice versa) in the prescribed order and at the correct times to ensure the validity of the output.

In particular, it is may be necessary to ensure that a process or transaction is neither omitted nor run more than once. The database of triples (UserID, TP_i, [CDI_a, CDI_b, CDI_c, ...]) proposed by Clark-Wilson can be implemented in such a way as to assure that a particular program must be run against a particular data item before another program can be run, but only if there is a unique intermediate output file CDI_i which cannot be output by any other TP.

Similarly, the triples mechanism is not sufficient in and of itself to protect against a program or data transaction being executed multiple times. A bank deposit could therefore be credited twice, or two checks issued in payment of a single invoice, without violating the Clark-Wilson TP/CDI information flow rules. In order to prevent this from occurring, either the data inputs must be tested and then marked by the processing program as having been processed, or the Access Control Lists themselves must be

updated to prevent a particular file from being accessed again.

In addition, running a program at an inappropriate time, *e.g.*, before the end of the banking day, or at night or on weekends when the ADP center is normally shut down, may result in incomplete or erroneous data and may also be indicative of an attempt to defraud. Any technique that can provide an assured pipeline can provide the necessary assurance of proper sequencing, but the TCB cannot reasonably be expected to build in all of the necessary time-of-day and day-of-week constraints and exceptions for each TP.

The use of the Clark-Wilson triples mechanism by itself is neither necessary nor sufficient to guarantee the proper *dynamic* performance of the system.

Finally, serious theoretical difficulties are involved in attempting to assure the general sequencing and synchronization of inputs and processes throughout a network, although these problems may be solvable in the context of a particular application.

It is therefore suggested that any requirement that TPs run in a certain sequence, or only at certain times of the day or days of the week, be addressed by the certifier of the TPs themselves. Instead, we will only provide the necessary mechanisms to allow the TPs to enforce these rules:

Rule 8 — Trusted Date/Time:

The TCB shall provide a trusted date/time function that is maintained to within N seconds of GMT, together with an offset or correction to obtain the local time, but which is not necessarily closely synchronized with any other TCB. Only the DAA or his designee shall be allowed to reset the date/time function, and any attempt to tamper with the date/time shall invalidate the digital signature of the TCB (by destroying the secret signature key of the DAA that is stored within the TCB), and thereby invalidate all data produced henceforth.

Rule 9 — Trusted Sequencing:

The TCB shall provide sequencing mechanisms that are sufficient to assure that programs are applied to data items in the prescribed order. The trusted Transformation Procedures themselves should be certified as implementing any necessary constraints as to the permitted times-of-day or days-of-the-week constraints on the execution of that TP.

3.10 Auditing and Journalling

Clark and Wilson suggest that all TPs write sufficient information to a protected data item (a transaction journal or audit log) to permit the reconstruction of the operation. Because the TCSEC requires that all TCBs of class C2 or higher maintain an audit log, an obvious question is whether the TCB-provided audit log would be sufficient for this purpose. Unfortunately, the answer is no.

3.10.1 Integrity Audit Requirements

The audit log required by the TCSEC is primarily concerned with detecting various threats that might lead to a compromise of sensitive or classified information: "The audit mechanism of a computer system has five important security goals. First, the audit mechanism must allow the review of patterns of access to individual objects, access histories of specific processes and individuals, and the use of the various protection mechanisms supported by the system and their effectiveness. Second, the audit mechanism must allow discovery of both users' and outsiders' repeated attempts to bypass the protection mechanisms. Third, the audit mechanism must allow discovery of any use of privileges that may occur when a user assumes a functionality with privileges greater than his or her own, *i.e.*, programmer or administrator. In this case there may be no bypass of security controls but nevertheless a violation is made possible. Fourth, the audit mechanism must act as a deterrent against perpetrators' habitual attempts to bypass the system protection mechanisms. The fifth goal of the audit mechanism is to supply an

additional form of **user assurance** that attempts to bypass the protection mechanisms are recorded and discovered. Even if the attempt to bypass the protection mechanism is successful, the audit trail will still provide assurance by its ability to aid in assessing the damage done by the violation, thus improving the system's ability to control the damage."[26]

The audit log required by the TCSEC is not very useful for integrity purposes, not only because insufficient information is recorded, but because the audit trail is considered sensitive, must be protected at the highest sensitivity level of the data contained on the system, and can only be accessed by audit personnel. The "auditors" (using the classical accounting meaning of the term) will probably not have the necessary clearances to review the TCB's audit log in a multilevel secure system, and the security administrators, on the other hand, may not have the training to investigate possible system integrity violations.

The TCSEC requirement for an audit log capability is introduced at the C2 level. Because a C2 system does not include mandatory access controls and is therefore not trusted to separate data by classification, the requirement to essentially classify the audit data as system-high makes sense. In addition, user login and other identity information may be sensitive, especially if erroneous passwords or userids are recorded, since that information might be used to guess the correct password. Finally, there are a number of high-bandwidth covert channel mechanisms that could be used to signal from a classified subject to a subject of lower classification via the audit log, if the audit log is not protected.

On the other hand, there is important integrity-relevant information which should be collected, even though it may have to be classified at the same level as the originating subject. If this problem cannot be overcome (perhaps by a trusted database approach, with the audit log entries being individually classified objects), then additional integrity audit logs may be required for each security classification level and category — an admittedly awkward approach. Because even the fact of existence of a classified object may itself be classified, it is necessary to restrict the access to a particular transaction (UserID, input objects, process, output objects) to the highest classification of any of the elements of the transaction. Unfortunately, in the case of a downgraded or "sanitized" object, the antecedents of the object may have to remain classified.

If all Transformation Procedures in fact always worked correctly, it could be argued that an integrity audit log would be unnecessary. Unfortunately, in even the best of systems sometimes things will go wrong due to human or mechanical errors, and it may be necessary to work backward to determine how a particular error occurred and how to correct it, or forward from a prior checkpoint by applying the transaction journal. In this regard, it should be noted that the Clark-Wilson database of triples is not constrained so that only one TP or one user could create a given CDI. Instead, any one of several TPs operating on behalf of any number of users could have conceivably created a given CDI, so if an error occurs the database of triples does not contain sufficient information to unambiguously pinpoint the source of the error.

From the standpoint of integrity, the fundamental audit requirement is to record which users ran what processes against what data inputs in order to produce a given output or outputs (including modification of the input.) The intent is to record an audit trail (not necessarily produced all at one time or in one place) which will clearly and unambiguously indicate how a particular object came to the current state, and to be able to derive how previous versions of that object looked at any point in the past. For that reason, financial transactions and processes involving strong configuration management should also keep a transaction journal which shows in greater detail exactly what operations were performed, so as to allow the data to be reconstructed or "backed-out" in the event of an error or required restart.

Because it may be important to detect a process which has been run twice, *e.g.*, printing an unauthorized set of duplicate payroll checks, it is necessary to collect all of this integrity-relevant information on a centralized basis (at least at each host).

---

26. **A Guide to Understanding Audit in Trusted Systems**, NCSC-TG-001, Version-2, National Computer Security Center, 1 June 1988.

3.10.2 Pedigree and Provenance

Although a centralized audit trail is necessary, it is not always sufficient. In particular, in a networking environment it will frequently be impractical to attempt to locate and access the appropriate audit database remotely. This is particularly true of data or programs that may be widely disseminated, where the transaction journal or audit log that contains the evidence of that object's creation may not be readily available to or known by the potential users of that program or data, or even all contained in one place.

For that reason, in addition to keeping a centralized audit log, **it would be highly desirable to incorporate the integrity-relevant audit data in a portion of an integrity label that is associated with the object (program or data item) itself**, and for a subject to be able to require that information be provided in order to establish how the current instance of that object (and previous instances, if necessary) was produced.

Therefore, in addition to the cryptographic checksum, attribution, and integrity domain markings contained in an integrity label, the audit data should optionally identify the process that was used to create the object; all the data inputs to that process; the date, time, and place of execution of the process; and the authorized user who initiated the process and/or approved the results. We will call such information in the object's integrity label the *pedigree* of the object. The pedigree should be cryptographically bound (*i.e.*, by a digital signature) to the object's contents so that there is no question as to its authenticity.

In many data processing applications, notably program development, there is a requirement for strong configuration management control over the final article. In order to ensure that the right components were included in the final "build," it is desirable to be able to work backwards from the pedigree of the final object to the ultimate source, *i.e.*, to produce a complete bill-of-materials. An example of the application of such a bill-of-materials would be in the generation of a trusted application program or the TCB itself, where it is necessary to clearly establish the identity of all of the base source program files plus all of the source program revisions (adds, modifies, and deletes); the macro libraries used; the compiler(s) used and the object modules produced; the link editor used and the link library referenced, plus the set of control statements used; etc.; so that an complete picture would be available of what made up that program or system.[27]

In addition to the strictly process-dependent derivative information that flows from the source code to the object modules to the load module and can therefore be traced backward, there is often additional ancillary information that is somewhat more loosely bound to the object. Examples would include the design specifications; user documentation; test specifications, test cases and results; performance specifications and final data, certification and accreditation information; restrictions and known problems; etc. In many cases the association of this ancillary information with the object occurs after the creation of the object and represents a result obtained *from* the object, *i.e.*, a forward linkage. Whereas the backward linkage of the pedigree can be expressed concisely and accurately in machine-readable form, this ancillary information typically involves an statement in English or some other natural language by an authorized user as to some fact or relationship.

William Murray has referred to the concept of ancillary information which is associated with an object as the *provenance* of the object. Although the Random House unabridged dictionary defines provenance as simply "the place or source of origin," the recent Supplement III to the definitive Oxford English Dictionary defines

---

27. Such a scheme was implemented in the configuration management controls of the SAFEGUARD anti-ballistic missile defense program produced by Bell Laboratories and IBM in the early 1970's. The original rationale for the scheme was to ensure that programs (which were loaded into read-only memory and therefore were compiled separately from data) were compiled or assembled after the data objects they referred to, in order to ensure that the proper offsets were used and also to ensure that recompilation of the program portion did not affect any program address constants stored in the data portion. The scheme was extended to carry the program identifiers of source and macro statements through to the object modules, and to carry the object module identifiers through to the "thread" tape that contained the final control program for the SAFEGUARD computer. Sufficient bill-of-materials information was therefore available to precisely define the entire contents of the system, together with the origin of each component. However, no cryptographic checksums were used to protect this information, so it was still possible to "patch" the program — a deplorable practice. The author would be interested to learn of other systems which have implemented such concepts.

provenance as "The history or pedigree of a work of art, manuscript, rare book, etc., *concretely*, a record of the ultimate derivation and passage of an item through its various owners." One of the references cited by the OED states "Also included would be all published documents, catalogues, and journals that contain references to the painting, along with reproductions, exhibitions, and sales records, as well as correspondence, especially of the artist, in which mention of it may be made." The term is used similarly in forestry to indicate the origin of a genetic line, and in etymology to include the derivation of a word together with illustrative examples of its use at a particular time. The use of this term for the present purpose therefore seems quite appropriate.

A recorded pedigree and/or provenance which is irrefutably bound to the contents of the object is a very powerful mechanism. Because it eliminates any doubt as to where, when, and how an object was created and what other objects or conditions were associated with it, it adds greatly to the concept of *closed loop control* of the system. The digital signature of the TCB will protect the pedigree from modification, but the provenance is essentially a statement by the object's originator and/or other parties *about* the object, and should therefore be digitally signed by those parties.

The pedigree and provenance of the object should be viewed as being part of the "container" of the object, and must therefore have the same sensitivity classification as the object itself so that it cannot be read by anyone who is not authorized access to the object.

3.10.3 Evidentiary Requirements

In the event that the audit trail discloses an illegal or unauthorized act by a user, it may be desirable to prosecute or otherwise punish the offender under due process of law. If the pedigree or audit log is to be used for such evidentiary purposes, it must satisfy two fundamental requirements:

- It must be possible to clearly and unambiguously establish what actions were performed by the offender, without having to resort to circumstantial or hearsay evidence. In particular, it must be possible to prove beyond a reasonable doubt that the offender did in fact commit the particular act that is alleged, and that no one could have possibly tampered with the evidence so as to shift the blame to an innocent party.
- In order to distinguish between a simple mistake and culpable negligence or malfeasance, it is usually necessary to show that the offender understood what he was doing, and that he showed willful and reckless disregard for the generally accepted standards and practices of his peers in the community. For that reason, it is often necessary to prove *intent*.

A digital signature or non-repudiation mechanism can be particularly helpful in satisfying both of these requirements.

Both the audit trail and the pedigree should be digitally signed by the TCB itself, acting as the agent of the DAA. Anti-tamper mechanisms should be in place which would prevent anyone from being able to substitute or modify the audit trial or forge the digital signature of the TCB without detection. If this is not possible, then it may be necessary to periodically print out and collect a hardcopy version of the audit trail in the routine course of doing business. In this case it would be desirable to have the user routinely review and manually sign or initial the audit record.

For transactions which are of considerable importance it would be highly desirable for the originating user (and/or other users, perhaps) to digitally sign them as part of the provenance. This would unambiguously indicate that the results have the user's approval and represent his willing and conscious act and are of his own volition. In this case the digital signature must be the user's own, and not just the digital signature of the TCB (which can properly attribute an object to the originating user, but cannot divine that user's intent). If the issue is recognized as being important, the receiving subject may require that the originator have signed the object indicating his approval before the object will be processed.[28]

The TCB must be trusted not to save and/or misapply the user's digital signature key without

his consent. Ideally, the user's digital signature should be computed on a "smart card" or similar device which the user possesses, so that the user's secret key is never out of his possession or control. Unfortunately, the available smart card technology is not yet capable of supporting a sufficiently long RSA key to provide adequate security with reasonable performance.

We therefore require the following auditing rules:

Rule 10 — Auditing:

In addition to the audit data which is required for security, the TCB shall at a minimum create an audit log entry for each *file* which is imported into the system or created or modified within the system. The audit log shall identify the authorized user responsible for initiating the execution of the process; the process that was executed; and the files and/or other objects (*e.g.*, telecommunications sessions) that were read by that process; together with the date, time, and the identity of the TCB on which the process executed. Each object shall be identified by its fully-qualified name, including the device ID and/or volume on which it resides together with the "path" or similar directory information, plus an ID string (cryptographic checksum) that uniquely represents the object's *contents*. Processes (programs that are executed by a particular user) must be similarly identified with the fully qualified program name including the device ID, volume ID, and "path," plus the cryptographic checksum of the program module that was executed. The originating user's full name (not his logon UserID, which may be assigned arbitrarily), plus his 128-bit unique ID string or cryptonym, shall be recorded. The identity and location of the TCB shall also be recorded. The date and time when the file was created (closed) shall be recorded, preferably in GMT together with time zone offset of local time from GMT, in order to avoid possible confusion as to time zones and local Daylight Saving Time practices. The information contained in the audit log (either separately or collectively) shall be classified at the same or higher level as the information from which it is derived. Only the TCB shall be able to write to the integrity audit log, and only the Security Administrator shall be able to read, reset, or erase the audit log.

Rule 11 — Pedigree:

The TCB shall provide the option to capture, in an integrity label that is permanently and irrefutably associated with that file, at least the audit log level of detail defined above (the pedigree) as it pertains to that file. The identity, location, and approving or controlling authority (DAA) of the TCB shall also be recorded in the integrity label so that the record can be traced back to the source if necessary. The originating TCB shall "sign" the pedigree, using a digital signature or other non-repudiation mechanism to ensure that the integrity of the label itself, as well as the association of the label with the contents of the file, cannot be compromised. The pedigree shall be considered part of the object, and therefore subject to the same mandatory and discretionary access controls as the data object itself.

Rule 12 — Provenance:

The TCB shall provide a means for a user to undeniably indicate his or her approval of the contents of a particular object and/or enter other

---

28. Some objects, *e.g.*, notably documents and forms which are circulated for comments and approval, may require a lower level of granularity than just the file. For example, only the originator of a document may be allowed to modify and/or sign a certain portion of a document. However, the custody of a document may be passed on to other users, who may add their comments, propose changes, and agree or disagree with the other comments. Finally, some comments may be incorporated and others deleted from the final document, with a few being collected to form a minority opinion which may be signed by one or more users. The overall system and available mechanisms should be flexible enough to accommodate such a process, but such specialized functions may be relegated to a database management system or an electronic colloquy system instead of the TCB.

data (typically in a natural language) to be associated with the object as the *provenance* of the object (*e.g.*, a description of the object's contents, associated documentation, etc.) The provenance shall be recorded in both the audit log and the integrity label of the object at the time it is generated or added to. Any authorized user (not just the originator) who can access the object can create or add to the provenance of an object and affix his unforgeable digital signature to it such that both that user's portion of the provenance and the contents of the object being signed are cryptographically bound together in such a manner that any change to either the provenance or the object will be detected. The non-repudiation mechanism shall require the explicit volition and consent of the signing user at the time it is applied, but at any time thereafter the cooperation of the user shall not be required in order to substantiate the fact of his prior approval. Any recipient of an object, including an independent third party, shall be able to establish the validity of the user's digital signature beyond any reasonable doubt. The provenance of an object shall be considered part of the object, and therefore subject to the same mandatory and discretionary access controls as the data object it pertains to.

3.11 Human-Readable Output

The previous discussion has concentrated on the problem of guaranteeing the integrity of computer-readable information, but we have not yet discussed the problem of getting the information into or out of the computer.

If we make use of cryptographic checksums attached to files, we can avoid having to trust the integrity of the network, storage devices, or even utility routines, since any modification of the information would be detected the next time the checksum was verified. The question is whether we have adequate mechanisms to protect human-readable input and output.

In both cases, we have what might be called the What You See Is What You Get (WYSIWYG) problem — how do we know that the information in a computer-readable file corresponds to what we see on the screen or on the printed page?

The basic problem, of course, is that most of the time what we see on the screen or on the printed page *doesn't* correspond directly to the contents of a computer file, because we only see the effects of a computer program operating on the data. Only a trusted hexadecimal dump program would show us exactly what is in a file, and for most purposes that would be impractical.

3.11.1 Trusted Input/Display

In order to limit the scope of the discussion somewhat, it will be assumed that all human-readable input will be typed in via a keyboard that is directly attached to a trusted host or workstation, so that the cryptographic checksums that are necessary to protect the machine-readable information from modification can be applied at the earliest possible moment. It will also be assumed that the TCB supports a trusted path to the user's video display, so that the TCB can reliably and unambiguously inform the user of the security and integrity levels associated with a session, for example.

In general, the problem with trusted input is not the input itself, but rather the correspondence between what is displayed on the video display and what is entered into the electronic record of the transaction. With the exception of passwords, which normally are *not* displayed on the screen, typing is not expected to be a high-integrity process. Instead, the human user is held responsible for reviewing and editing the input, and we assume that what is displayed on the screen after the input was edited is what was the user intended.

At first, this doesn't seem like too great a problem. Consider a TP that is used by a bank officer to certify payments. The TP would issue a call to the TCB to get the next character of input, display the character on the display appropriately, and perform the required function. Assuming that the TP is trusted for this particular application, what could go wrong?

From the standpoint of secrecy, we might worry whether another program, perhaps a corrupted version of a Terminate-and-Stay-Resident (TSR)

"pop-up" program such as SideKick, had been inserted between the keyboard processing program and the TP, perhaps by replacing the interrupt vector for the Get Next Character routine. Such programs are often used to activate special "hot key" functions and/or keyboard macros. However, assuming that we have a TCB of at least a B1 level of assurance, the keyboard will be connected to the subject that was created when the user logged on, and the mandatory access controls will prevent any such program from communicating any data from that subject to a subject of lower classification.

Even if the corrupted pop-up program were to add or delete keystrokes the data would still be displayed properly, and the user would be responsible for its correctness. Assuming that the TP is trusted to properly display whatever it reads and to take the appropriate action, we can conclude that no threat to the integrity of that TP could result from the modified keyboard input.

However, a TSR could conceivably modify the data that had been read in to the TP *after* it had been displayed. In particular, whenever some particular payee's name was entered, the program might wait until the *next* field was entered (after the information had been displayed on the screen but before it had been written out or checksummed), and then search memory for that payee's name and alter the amount to be paid. However, if the TCB implements an integrity containment mechanism (memory protection) which would prevent a low-integrity subject from modifying a high-integrity object this attack would be prevented.

A somewhat more plausible attack would involve inserting a TSR between the TP and the video display. For example, suppose that a corrupted version of a video driver program were installed. Such a program could conceivably be modified to search for a particular payee's name and if found delete the final digit of the amount to be paid. The user would believe that he mistyped the amount and would type in the "additional" digit, thereby multiplying the amount by ten (approximately). Another approach would be to simply fail to present on the screen all of the information that was read in from a file, thereby tricking the user into approving (and perhaps digitally signing) something that he never saw.

Corrupting a video display driver to attack a program or data may sound far-fetched, but it would not be at all difficult. In fact, some standard video drivers already allow such attacks as a system feature!

In MS-DOS, for example, the ANSI.SYS video display driver program can be installed to allow the user to conveniently change colors, manipulate the cursor, and control other aspects of the display using data that is embedded in the character stream that is to be displayed. If ANSI.SYS encounters an "escape" character (hexadecimal 1B) followed by a "[", the subsequent characters are interpreted as a command. Among the various video commands is the Set Graphics Rendition command, which can be used to change the graphics parameters, including the color of the foreground and/or background. In particular, if the string "ESC[8m" is sent to the screen in text mode (where ESC is the single escape character), **the text that follows will be displayed *invisibly*.** Although the use of the invisible printing mode is convenient when entering passwords, it also offers the opportunity for someone to include something in a text file which most editors and other programs would not display on the screen. Another ANSI sequence could turn the normal screen display attribute back on, and yet another could reset the cursor position. A slight modification to ANSI.SYS would allow the start of the invisible portion of the text to be signaled by a character or combination of characters that would not be printable by most printers, such as NUL (hexadecimal 00), or space-backspace. In this manner, the "invisible" text would not be displayed, but would be printed. Obviously all sorts of words like "not," "except," etc., could be inserted with grave damage to the meaning of the text with this technique, yet the file would appear to be properly checksummed.

At least one popular operating system fell victim to this type of manipulation several years ago, when users discovered that they could send invisible messages to other users. By including a command within the invisible portion of the message and leaving it on the user's screen, the next operating system command the user sent would include the invisible command just as though it had been sent by the user himself. The

command could export all of that user's data to the attacker, destroy or modify files, etc.

Other attacks could involve the character set that is loaded into the graphics controller card and/or the operating system. When operating in graphics mode, for example, MS-DOS uses software to define what character is to be drawn in response to a text character input, and this character table is unprotected in memory. By changing the representation of a character, for example "$" to "¥" or "£," or "+" to "–," various types of fraud could be carried out.

Similarly, most word processors, editors, and other MS-DOS programs will not process or display text that occurs after an End-Of-File character (Ctrl-Z or hexadecimal 1A) in the input text. As a result, an obvious security and/or integrity compromise may occur if a file is released that contains additional information not seen by the human reviewer, but which might be printed by a program that relies on the physical file length contained in the directory. In addition, word processors and editors frequently incorporate additional bits or characters in their output in order to represent such printing devices as superscripts, bold face, underlining, etc. Depending on the output mode selected, these characters may or may not appear on the screen, but they will appear in the output file unless a special "non-document mode" or ASCII output file is requested.

In order to deal with these problems, we must impose four conditions:

- Programs that interface to the user's display must either be certified as part of the TP itself (*i.e,* evaluated as part of the application, to whatever level of trust is required), or they must be a high-integrity trusted part of the TCB itself. This includes programs which define the character set and other aspects of the display, including a trusted reset of those attributes.
- If a display driver program is incorporated as part of the TCB, it must use explicitly defined escape sequences which involve control characters which are outside of the normal alphabet, and can therefore easily be detected by the TP.
- The TP must ensure that all output is displayed as intended, including scanning for escape sequences embedded in the data that might cause that data to be displayed in a misleading manner.
- The syntactic and semantic rules for a TP must ensure that any output created reflects the user's intent, particularly with respect to any information which was not displayed on the screen.

3.11.2 Trusted Output

In addition to trusted input, it is necessary to be concerned about the integrity of printed output. The problem of trusted printed output is substantially more difficult than the input and display problem, for several reasons:

First, we normally assume that typed input is unreliable, and therefore we scrutinize it carefully. Computer output, on the other hand, is normally assumed to be reliable, with the possible exception of page-formatting errors. For that reason, we tend not to examine printed output with great care, particularly if only minor changes are being made to a document that has been printed and reviewed previously.

Second, the processing required to transform a computer-readable file into a printed document is quite complex, especially if "desktop-publishing" quality of output is expected with the attendant use of proportional spacing, right margin justification, multiple columns, different fonts, superscripts and subscripts, and even embedded graphics. Although many programs offer a WYSIWYG capability, *i.e.,* a display on the screen that is as close as possible to what the printed output is supposed to look like, there is relatively little correspondence between what the document file looks like and what is sent to the printer.

Third, many newer printers are programmable in order to permit emulating other popular printers. The emulation programs are downloaded and put into operation by means of escape codes sent over the normal output channel. Unfortunately, once a new printer control program is loaded, there is generally no guaranteed reset sequence which would put the printer back into a secure operating mode, other than turning the power off. (The Centronics printer interface includes a

Reset line, but this may be interpreted by the printer control program instead of causing a reboot of the microcontroller.)

Finally, many printers allow software-defined fonts to be downloaded, even if the printer control program (*e.g.*, a PostScript interpreter) is kept in ROM and cannot be altered. The threat of some invisible piece of text changing "$" to "¥" or "£" would therefore exist.

Unfortunately, as is the case with compilers, sometimes it seems that just getting a word-processing program or print utility to function in accordance with its documentation is beyond the state of the art. Establishing the integrity of a program with respect to its normal word-processing function would therefore be difficult enough, and *proving* that it isn't sensitive to some combination of external inputs that would cause it to misbehave or do something else on command would be even more difficult.

In the case of the programmable printer, it would be quite easy to load a Trojan horse printer control program during a seemingly innocuous print job, so that it would then lie in wait for some later combination of data. Such a program could compromise both the secrecy and the integrity of the printed output:

- The secrecy of the output could be compromised by changing the mandatory labels of the output from say TOP SECRET to SECRET, or by storing the data and surreptitiously printing a second copy of the output.
- In addition to changing selected characters by redefining fonts, the integrity of the output could be compromised by printing selected records twice (*e.g.*, a check), or not at all (*e.g.*, a delinquent account notice), or by modifying the contents of certain fields (*e.g.*, a customer's account balance, to cover up an embezzlement).
- An integrity attack would not necessarily be limited to financial matters. Instead, if revenge were the primary motive, various "dirty tricks" might be carried out to damage an organization's reputation by printing misleading or inflammatory messages on the output.

In addition to these potential Trojan horse attacks, printers sometimes jam, and may either fail to print part of the output or in some cases may reprint a portion of the output during the recovery procedure. Page counts and/or prenumbered checks can help to ensure that the right amount of output was printed, but even then one page or check might be deleted and another printed twice without affecting the count.

There are several things that can be done to provide the greatest possible protection for printed output, depending on the threat, the vulnerabilities of the system, and the available resources:

- The printer should be trusted in both the security and integrity sense, *i.e.*, to print what it is supposed to and only what it is supposed to.
  - * The use of a printer control program that is contained in ROM and cannot be altered is preferred. Passive tamper-protection in the form of seals may be necessary to prevent or at least detect any attempt to modify the printer's control program.
  - * If the printer is programmable, the TCB should load a trusted printer control program and all of the necessary fonts before any high-integrity printing commences. Either the TCB or the installed printer control program must then block any attempt to download a new control program and/or fonts.
  - * If a trusted printer control program is not feasible, then the printer must be reset, if necessary by turning off the power and restarting it, before

any high-integrity output is printed.

- Only a trusted print program should be used to transform a high-integrity document file into a printed image. In particular, general purpose print programs, including word processors and print spoolers, should not be trusted to format and print a high-integrity file unless or until it has been extensively tested and is trusted with respect to the integrity domain of the CDI which is to be printed.

- In the absence of these measures, and preferably in addition to them, the user must be held responsible for reviewing the printed output visually, and ensuring that it agrees with the intent of the document file.

Obviously all of these concerns apply to plotters and other output devices as well.

3.11.3 Trusted Integrity Labels

Assuming that we are able to trust the output that is sent to the printer, perhaps because the TP outputs the characters directly, how can we indicate that this output resulted from a trusted process? In particular, how can we verify that an integrity label on a printed document is correct, and not a forgery? Although classified documents are required to contain the security classification of the document or page at the top and bottom of each page, there is no such requirement for integrity labels. Instead, the integrity of the output is usually indicated implicitly, although markings such as "DRAFT" may be used. For most documents, the most important integrity label is the signature of the person who is responsible. However, documents that are printed on corporate letterhead and other "official" forms are regarded as legally binding if used in the normal course of trade, even though they are not explicitly signed. An example would be a printed purchase order or invoice. Presumably it would be useful if we could provide a more accurate indication of the integrity of a document — something that would stand up in court at least as well as someone's signature or initials on a document.

For TCBs of class B1 or higher, the TCSEC requires that the beginning and end of all printed output must be marked with a banner page that correctly indicates the sensitivity of the output. The integrity level of the output could also be displayed on that page. However, as soon as that page is removed from the output the marking is lost, and of course anyone with access to a printer could duplicate that page.

The problem, then, is how to irrefutably bind the integrity label to the printed document in such a way that the accuracy of the label and its association with the document can easily be verified.

In the case of machine-printed checks we use a mechanical imprint device to print a signature, and notarized documents are embossed with a seal press. In both cases, however, the technology involved is trivial to duplicate, and we rely on external controls to minimize forgeries. Banks clearly cannot verify signatures on every check that is processed, so the responsibility for detecting a forgery is left to the customer when the monthly statement arrives.

Ideally, we would like to be able to directly calculate a cryptographic checksum and/or digital signature of a document from the computer-readable file, print the digital signature in an appendage to the document, and later verify the correctness of the document's contents by reading the document back in with an optical scanner and confirming the signature. In the case of a legal document, for example, this would obviate the need for the signers to initial every page of the printed document in order to detect the possible substitution of one or more pages. This would also protect against any changes caused by the word processor or print server, the printer, and even any erasures or alterations to the printed text itself.

Unfortunately, most word processors and print programs support page headings and footings, footnotes, subscripts and superscripts, multiple column output, floating section numbers and cross-references, a table of contents and index, etc., which rearrange or modify the text. In the general case, precomputing the checksum would require a complex program that would essentially replicate the function of the word processor or

print program, and would be almost as difficult to certify.

In the case of text-only documents, a compromise approach that would work reasonably well would be to create a "file" formatted document. Most word processors can prepare a version of a document which is suitable for the simplest kind of line printers, and then capture that output in a file. Such a document would not include any control codes other than carriage return/line feed and perhaps page eject, but all of the footnotes and other textual rearrangement would be performed properly. A cryptographic checksum program could then be run against that program, but all control codes and multiple spaces would be treated as a single space in order to eliminate line-length and other spacing differences. The digital signature of the resulting checksum could then be calculated and printed. The printed document could be validated by scanning the input using an optical scanner, or if necessary by retyping it, and then running the same checksum program against the result.

This approach has the drawback of requiring two steps — the creation of the "file" output and the checksumming of that output — and it still requires that we trust the word processor or print spooler. If the word processor has to be trusted in any case, we could eliminate the creation of the file output by having the TCB compute the checksum on the printer output stream after ignoring all control characters and compressing blanks, and then print the output upon receiving an escape character command that essentially says "print it here." Unfortunately, although PostScript printer sequences are bounded by delimiters, most other printer control sequences are open-ended and of variable length that are determined by context. The checksum program would therefore have to be sensitive to the particular control sequences used by the printer in order to interpret the output, which would greatly complicate the certification of the TCB program.

Another alternative would be to create the document first, have the user review and approve it, then scan the character text back in and compute the user's digital signature over the result. This signature would then be printed on a separate page to be attached to the original document. This would at least get the word processor or print spooler out of the loop, but it would mean that the scanning and signing program would have to be trusted.

In any case, a text-only approach cannot cope with ~~strike-outs~~, underlining, overprinting ($\hat{x}$), or subscripts or superscripts like H_2O and $E = MC^2$. Even tabular data (*e.g.*, a features list that places an X in the appropriate column), cannot be handled properly unless blanks are significant. And obviously **different** *fonts* and ***type*** faces, various sizes of print, and embedded graphics cannot be handled by this approach at all.

What is really necessary is a signature scheme that would consider a document as a series of page images. One approach would be to digitize the printed image as though it were a facsimile transmission, including every pixel on the page. Unfortunately, such an approach would be very dependent on the resolution of the scanner, and might also be thrown off by incidental smudges or specks of dirt. Slightly different magnification ratios, skew, blur, and distortion would also cause problems.

As a result, a standard pass/fail cryptographic checksum technique could not be used, and instead a "fuzzy checksum" or template technique would have to be used that could accommodate a certain amount of error. Perhaps a two-dimensional Fourier transform could be computed, and the higher frequencies thrown away to minimize the effects of noise. Because the resulting coefficients or template might be quite lengthy, it would be desirable to compute a checksum of the template, and then digitally sign the checksum. When the document was to be verified, the template would be regenerated and checksummed, and the new checksum compared to the digitally signed version. If the problems of magnification, skew, and distortion can be minimized, the low-frequency components of the template should be the same, and therefore the checksums should agree.[29] In addition, we have the problem of differentiating between the original and a copy of a document. Until recently, the differences between the printing and copying technologies were such that it was

29. Unfortunately, periods and commas are syntactically very important ($1.00 is not the same as $100), and the size of a period in a footnote may be smaller than some dirt specks.

relatively easy to distinguish between a copy of a document and the document itself. However, as these technologies begin to converge, it is increasingly possible to produce a copy of a document which is indistinguishable from the original.

We would therefore like to be able to indisputably validate the *contents* of a document, including a copy of a document, but still be able to reject *forgeries* or unauthorized duplicates of documents which are inherently valuable, *e.g.*, stocks, bonds, currency, receipts, etc.

Assuming that we can authenticate the contents of the original document it would not be difficult to validate the contents of a reasonably good copy. Authenticating a unique document (the container, not just the contents) is much more difficult, however, because it requires an irreproducible token be associated with the document itself, and irreproducibility is a function of the technology and resources available. One approach that has been proposed[30] would be to make use of the inherently random aspects of the paper grain or fiber itself, scanning the fibers within a carefully delimited area with a high-resolution, contrast-enhancing microdensitometer, and then incorporating the result into the digital signature along with the checksum of the contents of the document.[31] Another technique might make use of a random dispersion of magnetic particles or wires embedded in the paper or other support medium which could be scanned magnetically, or embedding short pieces of fiber optics in the paper in order to cause scintillation to occur at other spots on the paper whenever one end of the light pipe was illuminated.[32]

All of these techniques are subject to the problem of noise and various systematic variations. An exact comparison cannot be made — instead it is necessary to settle for some fraction of the bits being in agreement, with a sufficient number of bits being used to make a random coincidence extremely unlikely.

Having said all of this, we must reluctantly conclude that as yet there isn't any good, practical way of automatically authenticating and validating the integrity label or the contents of a printed document, and leave it as a question for further research. Until such time as an automated system can be produced, procedural methods and manual signatures will probably continue to be required.

We can therefore conclude this section by stating the following rule, based on today's technology:

Rule 13 — Human-Readable Output:

Transformation Procedures shall be certified to correctly display and print the desired human-readable output. To the extent that other programs may be interposed between the TP and the video display or printed page, those programs shall either be evaluated and certified as part of the TP certification, or they shall be a part of the Trusted Computing Base. Failing that, the human user must be held responsible for the integrity of the printed output.

---

30. I believe this suggestion was made by David Chaum, but I have not been able to track down a reference.

31. When an exceptionally valuable book or other paper document has sustained damage (holes or torn pages), the Library of Congress repairs such damage by carefully analyzing the constituents of the paper (wood fibers, flax, linen, cotton, etc.), then making up a slurry of those ingredients in water. The book is taken apart, and after measuring the size of the hole the damaged page is placed on a wire screen of exactly the right size. The appropriate amount of slurry is then poured on the page, and the mixture is sucked through the hole by a pump. The water is extracted, but the slurry mixture exactly fills the hole and is held in place by the screen. After the page is dried, the repair is virtually invisible, for that is essentially the way the paper was made in the first place. A variation on this technique could perhaps be used to cut and paste the scanned portion of the document onto/into a duplicate, but this would destroy the original. Although a manual signature could perhaps be moved to a spurious document this way, incorporating the scanned portion in a digital signature would prevent the contents of the object from being altered without detection.

32. Conversation with Dr. Augustus Simmons, Sandia National Laboratory, 24 January 1989, regarding feasibility studies for new ways of protecting currency against forgery.

4 POLICIES AND MECHANISMS

The previous section defined a set of rules in the spirit of Clark-Wilson which are claimed to be both necessary and sufficient to satisfy the military and commercial integrity concerns in a networking environment. A set of implementation mechanisms will now be proposed which will satisfy the above rules, not only in the restricted case of a single, isolated TCB, but in the more general case of a large network of interconnected TCBs, not all of whom necessarily share exactly the same security and integrity policies. This will be accomplished through the imposition of a label on each file which is stored or processed within the TCB, which will include a digital signature and integrity markings sufficient to support a mandatory integrity control policy plus a discretionary integrity control policy, as will now be defined.

4.1 Integrity Labels

The concept of a *file* as a collection of objects is not required of a TCB by the TCSEC, and in fact it could be argued that the implementation of such a mechanism should be excluded from the security-critical kernel of the TCB in accordance with the notion of least privilege. Nonetheless, virtually all operating systems implement some form of aggregation of objects so that they can be catalogued and manipulated as a more convenient unit. Although lower level objects, *e.g.*, records, are sometimes used as the basis for mandatory access control *within* TCBs, the file is the most general and most convenient means of interchanging data *between* TCBs.

We would therefore like to have a means of exchanging files between users and TCBs that would ensure both security and integrity, no matter what form the file might take. In particular, the interchange should support a *virtual network* that may range from a hand-carried floppy disk to a satellite link to a high-speed fiber-optic LAN. In addition, it would be desirable if we could support files across a variety of formats and operating system architectures.

In general, we may assume that the files will be encrypted for secrecy, or at least authenticated, and therefore it will be necessary to pass the file through an encryption/decryption/validation program as it is being read or written, both to make it intelligible and to validate it. Therefore, rather than worrying about how to squeeze all of the necessary information into existing operating system labels (which almost surely do not have sufficient space reserved, and are generally incompatible across operating systems), we can incorporate the necessary security and integrity information as header and trailer labels within the encrypted file itself, in a format that only the file encryption/decryption program needs to understand. In addition, we may choose to implement some form of data compression, which has to take place *after* the file is authenticated (so as to include the potentially low-integrity compression algorithm in the file validation process), but *before* encryption (since encrypted information is essentially random, and cannot be compressed.) Again, only the encryption/decryption routine would be involved in the compression, so there would be no impact on the operating system code.

Depending on our security, integrity, and performance requirements, we may require that all files, even temporary virtual-disk files and "pipes," be maintained in this format. Or we may only require that those files which are stored on permanent media or transported outside the TCB be encrypted and authenticated. Temporary files, in particular the work files used by a single program such as a sort program or compiler, or those that are confined to a "workspace" used by only one user, might be exempted from these requirements in the interest of performance if there are sufficient protective measures to ensure that an active attack against these files could not take place while a program was running.

Obviously the file label should include the mandatory and discretionary access control information that pertains to the file. In particular, it is assumed that the file label contains the classification level of the *decrypted* contents of the file, together with the inclusive and exclusive discretionary access control attributes necessary to properly control the file once it is decrypted.[33]

The integrity label will contain a cryptographic checksum of the entire contents of the file, unless the RECORD attribute is specified as discussed below. The name and unique cryptonym of the originator will also be included, in order to satisfy Rules 1 and 2 (Immutability and Attribution). The identity and authorization of the user will be established through a bottom-up system of digitally signed certificates which the user can carry with him or transport over the network, as was discussed in the previous section.

At a minimum, a Message Authentication Code or an encrypted Manipulation Detection Code will protect the fields within the security/integrity file label from undetected modification when the files are contained within the TCB. The digital signature of the TCB can be applied to the file label when the file is to be exported (transmitted or stored) outside of the TCB, *e.g.,* on a floppy disk or on a LAN-based file server, or optionally for all files all of the time. The authenticity of the TCB's digital signature will be confirmed by a certificate which is signed by the DAA. The public signature key of the DAA will be securely installed within the TCB, and protected by a cryptographic tamper-variable.[34]

In addition, the necessary markings for a label-based Mandatory Integrity Control policy will be provided, as will be discussed below. The information necessary to implement a Discretionary Integrity Control policy will also be provided. Finally, an enhanced audit capability will be provided through the use of pedigree and provenance features.

The proposed Mandatory Integrity Control policy that will govern the transfer of information between subjects, whether within a TCB or across the network, will now be discussed.

4.2 Mandatory Integrity Controls

By far the most often referenced model for integrity is Biba's integrity model,[35] which is basically the dual of the Bell and LaPadula security policy model. The Biba model imposes controls to prevent high integrity subjects from drawing a false inference as a result of "reading-down" a low integrity object (simple integrity property), and to prevent a low integrity subject from corrupting or "writing-up" to a high integrity object (integrity confinement property or *-property).

The Biba integrity policy model has been criticized as unworkable, both because it allegedly does not provide for untrusted data input,[36] and because it allegedly requires an "excessive" degree of trust in the executing programs.[37] Those criticisms are now widely recognized to be overstated.

It is true that the simplest case of a hierarchical integrity policy, one where both the subject and the object are constrained to one integrity level and there are no integrity categories, is likely to give rise to serious operational difficulties. However, if a subject is described as being trusted to operate within a range of integrity levels and more particularly integrity categories, as

---

33. Because even an A1 TCB is not totally trusted, the encrypted file may have a classification level that represents the minimum that TCB is allowed to output. For example, a B2 level system operating in a closed environment is considered sufficient to handle information which ranges from Secret to Unclassified, *or* from Top Secret to Confidential. If Top Secret data is encrypted under the control of a B2 TCB, the minimum classification level of the output is Confidential, and all systems which process that data would have to protect the information at that level. Only a TCB which possesses the cryptographic keys necessary to decrypt the file would be able to read the information, and after decryption the file's contents would have to be processed as Top Secret again.

34. Particularly in the case of a portable or "laptop" TCB, or one which the user is not sure is trustworthy, it is advisable that the identification and authentication process that occurs at logon operate in both directions. That is, the TCB should provide unforgeable proof of its own *bona fides* to the user at the same time that the user's identity and authorization is being confirmed to the TCB, without revealing any secret information in either direction before the TCB and the user are mutually satisfied. This can be accomplished by a challenge and response process, whereby the user enters a challenge phrase (via floppy disk or smart card) which only he knows, encrypted in the public key of the DAA. The TCB decrypts the challenge phrase using the secret key loaded by the DAA, and presents it to the user. The user then recognizes the challenge, and can conclude that the DAA must have accredited the TCB and that no tampering has taken place (otherwise the secret key would have been zeroized by the TCB's tamper-detecting mechanism.) The user can then enter his normal response password or passphrase, which is used to decrypt the rest of the information on the smart card or floppy disk and thereby confirm his identity to the TCB. Only the DAA has to be trusted for this mechanism to work.

35. Biba, K. J., "Integrity Considerations for Secure Computer Systems," Mitre TR-3153, Mitre Corp., Bedford, Mass, April 1977.

36. Clark and Wilson, *ibid.*.

37. Boebert, W. E., and R. Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," **Proceedings of the 8th National Computer Security Conference**, 30 September - 3 October 1985, Gaithersburg, MD.

described by Shirley and Schell,[38] Schell and Denning,[39] Lee,[40] Karger,[41] Shockley,[42] and the author,[43] the alleged difficulties go away immediately.

The Biba model of integrity control (*i.e.*, the simple integrity property and the integrity confinement property or *-property) can therefore be extended through the implementation of both minimum and maximum integrity markings associated with the subject, where the minimum and the maximum integrity markings have both a hierarchical and a non-hierarchical component.

Finally, in addition to preventing a low-integrity subject from creating or modifying higher-integrity data, **it is desirable (but not always possible) to prevent a low-integrity subject from deleting or renaming a higher-integrity object.**

It is generally very difficult if not impossible to detect the deletion of an object unless that object is somehow chained to another object that is not deleted. In addition, it is difficult or impossible to prevent such a deletion if the object is stored or transmitted outside of the TCB or trusted network. This situation is therefore different from the other integrity problems that we have dealt with. Admittedly, it could be argued that the deletion of an object could result in a low-integrity subject misleading a high-integrity subject into believing that the deleted object never existed, and that therefore this is an valid integrity problem. However, a much more conservative way to design the high-integrity program would be to use an object containing an explicit null (such as an end-of-file character) as a signal, rather than rely on the absence of an object to indicate that no input was intended. With such a design a "File Not Found" error will immediately provide a indication that something that is wrong, and the standard actions can be taken to restore the object from the archive or take remedial action to recreate it.

The unauthorized deletion or renaming of an object will therefore be considered a significant nuisance, but a denial-of-service issue rather than an integrity problem.

The mandatory integrity requirement (Rule 3) can therefore be restated as follows, given a multilevel integrity policy for subjects:

1. A subject may only access, accept, or believe an object if the integrity-write-limit marking of the object dominates the integrity-read-limit of the subject; *i.e.*, a subject cannot read down in integrity below a specified minimum.

2. A subject may not assign an integrity-write-limit marking to an object that it creates such that the integrity-write-limit of the subject would be dominated by the integrity-write-limit marking of the object; *i.e.*, a subject cannot write up in integrity beyond its specified maximum.[44]

3. A subject within a TCB may not delete an existing object unless the integrity-write-limit of the subject dominates the integrity-write-limit marking of the object to be deleted.

---

38. Shirley, L. J. and R. R. Schell, "Mechanism Sufficiency Validation by Assignment," **Proceedings of the 1981 IEEE Symposium on Security and Privacy**, IEEE Computer Society Press, Washington, D.C., April 1981, pp 26-32.

39. Schell, R. R. and D. E. Denning, "Integrity in Trusted Database Systems," **Proceedings of the 9th National Computer Security Conference**, NBS/NSA, 15-18 September 1986, pp 30-36.

40. Lee, T. M. P., "Using Mandatory Integrity To Enforce "Commercial" Security," **Proceedings of the 1988 IEEE Symposium on Security and Privacy**, 18-21 April 1988, Oakland, CA; IEEE Computer Society Press, Washington, D.C., p. 140-146.

41. Karger, Paul, "Implementing Commercial Data Integrity with Secure Capabilities," **Proceedings of the 1988 IEEE Symposium on Security and Privacy**, April 18-21, 1988, Oakland, CA; IEEE Computer Society Press, Washington, DC, 1988, pp 130-139.

42. Shockley, W. R., "Implementing the Clark/Wilson Integrity Policy Using Current Technology," **Proceedings of the 11th National Computer Security Conference**, NBS/NSA, 17-20 October 1988, pp. 29-37.

43. Jueneman, R. R., "End-To-End Compromise and Integrity Controls," Computer Sciences Corp. Technical Report CSC-TR/87-3001, May 1987.

44. A subject may assign an output object an integrity marking that is lower than (dominated by) its own integrity-read-limit. An example would be the case when a catastrophic error such as a divide by zero exception destroys the meaning of a particular calculation, yet it is necessary to post some form of result as an indication of abnormal termination. The subject might not then be allowed to reread or believe that object, which is the intended result.

4. Modification, including rename, will be considered equivalent to reading the existing object (if necessary), creating a new or modified instance of that object with the same or different name, and deleting the previous object. For that reason, an object may not be modified or renamed unless the integrity-write-limit of the subject dominates the integrity-write-limit marking of the existing object, *and* the integrity-write-limit of the modified or renamed object is dominated by the integrity-write-limit of the subject.[45]

5. If the cryptographic checksum or digital signature verification fails when an object is accessed, either the object or the checksum has been presumably been modified by an unauthorized user or process. The integrity of such an object shall be set to null, *i.e.*, given a hierarchical integrity level of 0 and given a null set of non-hierarchical integrity categories, indicating that it is an Untrusted Data Item (UDI).

A subject that is trusted to operate over multiple integrity levels and/or categories corresponds directly to a Clark and Wilson TP that processes a UDI, transforming it into a CDI. Such a process does require trust, and it does have to be certified to perform its function correctly; but that is exactly what application processes do routinely and there is no reason why such a mechanism should be rejected. Indeed, the *raison-d'etre* of almost all data processing applications is to increase the integrity of the final result.

However, it is most important to realize that just because a subject is trusted to operate over a range of *integrity* levels does not mean that it has to be trusted in the multilevel *security* sense, *i.e.*, that it must be trusted to violate the Bell-LaPadula mandatory security constraints.

The fundamental appeal of the Biba integrity policy is that it can easily be modeled and implemented through a system of cryptographically-sealed labels and is therefore suitable for a network implementation. In addition, as Shockley points out, by implementing *readclass* and *writeclass* mechanisms the way that the Gemini Computers' GEMSOS TCB does, the Bell-LaPadula and Biba models can be combined into a common lattice representation, and a common mechanism can be used to enforce both the mandatory security policy and the integrity policy with a very high degree of assurance. This would obviously be very desirable, for it would implement the integrity confinement mechanism within the highly-trusted security kernel of the system.

It therefore appears that the requirement to constrain TPs and CDIs to an assured pipeline can be implemented through the use of Biba model, using integrity "categories" that are restricted to appropriately authorized data originators in the same way that the Bell and LaPadula categories are used to restrict information to appropriately cleared data receivers.

The difference is that in the Bell and LaPadula access control model, a subject cannot access (read or execute) certain categories of information unless the *subject* possesses the appropriate hierarchical sensitivity level and non-hierarchical sensitivity categories, whereas in the Biba integrity control model the subject cannot accept (believe) certain information unless the *object* possesses the appropriate hierarchical integrity level and non-hierarchical integrity categories, derived in turn from duly authorized subjects.

One possible use of a multilevel integrity-trusted subject mechanism would be to filter objects through syntactic and semantic tests prior to allowing other subjects to accept (believe) them, using the assured pipeline approach. Any program such as an editor, compiler, spelling checker, or other syntactic or semantic processor can have as its primary purpose the automatic scanning of input data and the implicit or explicit upgrading or improvement in the "quality" of the data, against whatever integrity domain is desired.

If a simple pass-fail integrity criteria is desired, as suggested by the Clark and Wilson approach to trusted Transformation Procedures (TPs), then the minimum and maximum hierarchical integrity levels can be set equal to each other, and integrity

45. A trusted subject should normally reject a transaction rather than replacing an existing object with an object of a lower integrity, but the deliberate downgrading of integrity is allowed and does not constitute a denial-of-service attack.

categories alone could be used to control read and write access, as proposed by both Shockley and Lee. On the other hand, if no integrity constraints are desired, then the minimum and maximum hierarchical levels of the subject can be set to system-low and system-high, respectively, without any non-hierarchical integrity categories being specified for the integrity-read-limit, and with "ALL" categories being specified for the integrity-write-limit.

A much finer degree of control than the strictly hierarchical Biba integrity model would permit can be achieved by allowing a multilevel integrity-trusted subject to specify the integrity level of some object to any appropriate value, within the integrity-write-limit constraint of the subject itself. Whereas the straight Biba model without multilevel subjects essentially implements a No Garbage In, No Garbage Out policy, other policies would allow Garbage In, Wisdom Out (validity checking and correcting data input, or statistical inference or averaging) or even Wisdom In, Garbage Out (anti-aggregation, or Census-type privacy, perhaps by deliberately adding noise to existing data so as to lower the precision.[46]

It should be pointed out that Garbage In, Garbage Out is an example of a high integrity process *if* a high integrity subject which accesses a low integrity object sets the integrity of the resulting object properly.

4.3 Hierarchical and Non-hierarchical Integrity Components

As discussed previously, the integrity of a process, and therefore the integrity of any resulting data, can only be assessed in relation to a set of defined syntactic and semantic rules. Since it is impossible to define the complete syntactic and semantic rules for the entire universe, the integrity of a process and any results it produces must be evaluated against a subset of the universe of all possible rules, *i.e.*, against a defined **integrity domain**.

Consider the case of a spelling checker, for example. If it encounters a word such as "tungue," it may suggest the replacement "tongue." If we pass a document through such a spelling checker and it finds no such errors, we have some confidence that there are no grossly misspelled words, and if that is the metric for integrity in this case, we are entitled to assign the modified output a higher integrity level than the input data or to regrade or promote integrity level of the original data. But what if the document were to contain some of the more awkward constructs of English orthography? Depending on the techniques employed and the preference of the program's creator(s), most spelling checkers would reject "deers," might or might not accept "memorandums" instead of "memoranda," and most would probably accept "mother-in-laws" instead of "mothers-in-law." The best spelling checkers might suggest that "tungue" could be either "tung" or "tongue," depending on the context.

For that reason, we must put an upper limit on the integrity that we will allow a spelling checker to assign to a document, based on an expert assessment of the extent to which the spelling checker "correctly" implements the rules of English spelling. It should be re-emphasized that the integrity of the object is not inherent in the object itself, but only when it is evaluated against the externally-defined set of *syntactic and semantic rules* that make up the integrity domain.

As an example of the applicability of these concepts, a major law firm might use a variety of such spelling checkers, some of which would use low-integrity but very fast rule-based algorithms that would reject "trasnposition" but might accept "memorandums," while a high-integrity product might contain the complete Oxford English Dictionary. Since even the OED-based checker would probably not catch an error such as "Deer Mr. Smith," an experienced human editor would have to be the penultimate authority, with the final draft being returned to the author to ensure that the sense of the

46. It is generally assumed that the objective of a system of integrity controls is to assure the highest possible integrity of the data, but privacy-supporting systems may limit information disclosure to a specified *maximum* integrity level. There is an interesting cross-over into access control mechanisms at this point, for highly precise data may have to be more highly classified so as not to reveal an intelligence-gathering capability or specific system capabilities. It is possible that an integrity control mechanism could provide a more tractable approach to this inference control problem than presently exists.

document had not been compromised. The firm might therefore have a rule that would require that all interoffice correspondence be spell-checked to at least a level of "2" by the low-integrity checker, while contracts, wills, and other legal documents would have to be checked to a level of "5" before being sent out.

It is conceivable that a draft could receive a "2" from the rule-based checker, yet upon being checked by the Oxford checker the integrity as it stood would be lowered to "0." If the Oxford checker were allowed to make corrections according to its rules, an integrity level of "4" might be assigned, depending on the degree to which it finds context-sensitive situations which it cannot resolve. After that, the human editor could review and/or edit the document, after which it could be assigned an integrity level of "5." Finally, the author would be asked to approve the document by digitally signing it, in which case the integrity level would remain at "5," or else he could make corrections, in which case the integrity would revert back to "0" since new spelling errors might have been introduced. Once the document was assigned an integrity level of greater than "2," the rule-based checker would not be allowed to modify it; and once the human editor had reviewed it and assigned it a level of "5" even the Oxford checker would not be allowed to modify it.

It might be convenient to be able to assign verbal labels to the various hierarchical integrity levels that would correspond to the sensitivity labels of UNCLASSIFIED through TOP SECRET, in the sense of "God said it," "The Pope said it," "The Bishop said it," "The Sunday School teacher said it," "It was written on the bathroom wall," etc. But that would necessitate much discussion about the nature of trust, who determines it, what domain is being referred to, how much damage is "exceptionally grave damage," etc. Similar objections apply to the use of HIGHLY-CRITICAL, CRITICAL, or NONCRITICAL as mandated by AFR 205-16, since there is no objective determination of how critical is CRITICAL, and because the same program or data may be HIGHLY-CRITICAL for one mission and NONCRITICAL for another mission since criticality is a function of the mission requirements and not the information.

However, although the determination of the classification level of an object is inherently subjective and may change or be changed as a function of time (either because the information has become known, or because it has become irrelevant), there are objective ways of defining and measuring the integrity of a process and therefore the integrity of a data object produced by that process, and that integrity should not change unless and accidental or deliberate modification to the object has occurred, or unless the external reality has changed.

For that reason, the hierarchical integrity associated with an object is defined to be the estimated probability that the process which created that object did so in accordance with the defined syntactic and semantic rules of a particular integrity domain or domains, either as estimated by an expert evaluator or obtained though objective measurement. The hierarchical integrity component is therefore an indicator of the *assurance* associated with the evaluation of a process against a specified integrity domain.

A convenient way of expressing the hierarchical integrity level would be to compute the number of leading nines in the decimal fraction of the estimated probability, computed as floor[$-\log(-\log(P))$]. For example, if a process were expected to fail on a random basis once in every million executions, the probability that the process conforms to the rules would be .999999. Therefore $-\log(-\log(.999999)) = -\log(4.34295\text{x}10^{-7})] = 6.36216$, and after truncation the result is 6.

With this scheme, a range of probabilities from less than .9 to .999999999999999 (0 to 15 nines) can be expressed within one 4-bit hierarchical integrity field.

This application of a hierarchical integrity policy with an objective measure for the hierarchical integrity level overcomes the previous objections to the Biba model and makes good intuitive sense for a single integrity domain such as spelling accuracy. However, suppose that we run the targeting data for an MX missile through the spelling checker, even the Oxford dictionary-based high-integrity version. Other than informing us that "Peking" is now to be spelled

"Beijing," what does a hierarchical integrity rating of "5" with respect to spelling accuracy tell us about the level of confidence we should have in the target data? Obviously, very little.

Unfortunately, many of the discussions of integrity in the past have been muddled because the authors talked about integrity as through it were some kind of universal truth that would apply to all programs or data.

The problem is clearly one of specifying the correct integrity domain for the required purpose. Spelling accuracy cannot be evaluated by a missile target validation program, and missile target data cannot be validated by a spelling checker. If we wish to quantitatively state the hierarchical integrity that is assigned to an object, it is necessary to indicate which integrity domain was used during the evaluation process.

We can therefore qualify the hierarchical integrity level of a subject or object with a *non-hierarchical integrity category*, letting the name or other representation of the integrity category be a shorthand way of identifying the syntactic and semantic rules of the integrity domain that applied to that subject or object.

Eventually, we will presumably begin to build up a set of heuristics that will allow someone who certifies a program to estimate the probability that the program conforms to the rules of the integrity domain. In the case of a TCB, for example, perhaps a C1 level of assurance would correspond to a hierarchical rating of 4, a C2 TCB would have a rating of 5, a B1 TCB would have a rating of 6, a B2 TCB a rating of 7, a B3 TCB a rating of 10 (because there is a substantial difference in the assurance requirements between a B2 and a B3 system), and an A1 system would have a rating of 12. The point is that it is not so much the absolute accuracy of the hierarchical integrity component that is important, as it is the relative comparison between two objects that have the same integrity domain.

Assuming that a consistent set of evaluation criteria is established for applications, the DAA or other approving authority for an application can easily specify whatever probability or assurance is must be provided in order to meet the requirements for a CRITICAL or HIGHLY-CRITICAL process.

In the example we used previously, if we want to determine whether a contract is ready to be signed, we should insist that a given object be rated a "5" or higher *against a specified integrity domain*, in this case spelling accuracy. If we want to be absolutely sure that the missiles are targeted for the Kwajalein atoll in the Pacific missile test range and not Honolulu, we should require a very high hierarchical integrity level be achieved against the targeting data integrity domain before the missiles are released.

If we need to evaluate some data object against two or more integrity domains, for example for both spelling accuracy and for target data accuracy, we could simply run the data through two different processes serially and combine the results somehow, but only if the evaluation processes do not change the input data but only evaluate it.[47] Since each evaluation process has only been certified with respect to the particular integrity domain, we cannot legitimately assume anything at all about the process with respect to any other criteria, especially if either process can modify the data. However, if the evaluation processes do not modify the data, the TCB could be trusted to correctly combine the results of two or more independent evaluations in accordance with a fixed rule.

In the case of the non-hierarchical integrity categories which represent the evaluated integrity domain, the rule for combining evaluations is simple — the result is the union of the previously existing integrity categories with the newly evaluated category.

The hierarchical or numeric ratings can be combined in accordance with the laws of

---

47. We might hope that an evaluation program would refrain from changing any data outside of its field of competence, *i.e.* its evaluated integrity domain, but in the past the author has suffered from an overzealous (and uninformed) human editor assigned to "clean up" the first draft of an IBM Technical Reference Manual, who changed a reference to an OS/360 JOB card to an "Employment Reference Card"; had a secretary transcribing dictation who didn't know the word "compilation" and so spelled it "copulation"; and most recently witnessed the result of a spelling checker program which automatically changed "DEC" (referring to the computer manufacturer) to "December" in an executive-level presentation. *According to their standards*, these were all well-intentioned changes.

compound probability, assuming that the probability of failure against one integrity domain is independent of the probability of failure against another domain. Thus, if I_1 is the hierarchical integrity evaluation with respect to category 1, and I_2 is the hierarchical integrity evaluation with respect to category 2, then the joint hierarchical integrity I_{12} is

$$I_{12} = \text{floor}[-\log(1 - [1-10^{-I_1}]\,[1-10^{-I_2}])]$$

For example, if the probability that there is a misspelled word in the target data is one in a million, then the probability that there is no misspelled word is .999999, and $I_1 = 6$. If the probability that all of the missile targets are correct (whatever "correct" is defined to mean) is .999, then $I_2 = 3$. Since .999999 x .999 = .998999, if we assume that the probabilities are independent we would expect the joint integrity (*i.e.*, that all of the words are spelled correctly *and* all of the targets are correct) to be 2. Applying the formula, $I_{12} = \text{floor}[-\log([1 - 1-10^{-6}]\,[1-10^{-3}])] = \text{floor}[2.99] = 2$.

Although there will be some loss of precision with this process if a number of integrity categories are evaluated sequentially, the calculations are conservative and the alternative of using floating point notation is not worth the complexity, especially since the original probability estimates may not have been all that precise in any case.

Finally, as we discussed earlier, even if the hardware, the operating system, the application programs, and all of the human inputs to a system are perfect, there may still be a divergence between the computer model and the external real world if the real world changes and the model does not. In order to prevent this from happening, it is necessary to periodically run an Initial Verification Procedure (IVP) which performs a reconciliation between the computer model and the external world. Although it may not be possible to predict when the external reality may change (when someone will steal something from inventory, for example), we can set whatever limit we like on how often the reconciliation must take place by *downgrading* the hierarchical integrity of a CDI after a certain interval, or at a specific date/time.[48]

We will therefore assume that the integrity label will contain a field that says essentially, "decrease the hierarchical integrity value of this object by X every T seconds," or alternately, "decrease the hierarchical integrity value to Y at date/time D."

4.4 Reference Monitor Implementation

It is perhaps worthwhile to stop and point out the value of the integrity containment mechanism that has been developed so far.

Consider a simple utility program such as a Move/Copy or disk compaction program. If such a program is to be at all useful, it must be trusted to correctly move files around on the disk regardless of their contents or the various integrity categories that the objects being moved or copied might have. However, if our concept of the TCB as providing an integrity containment mechanism is valid, we would hope that all of the various utilities furnished with an operating system would *not* have to be trusted in the sense of being formally evaluated and certified, but could instead be unevaluated, low-integrity programs. The problem is that a low integrity subject could move or copy a high integrity object, but the resulting object would be marked as having low integrity, thereby presumably destroying the value of the moved or copied object.

There is a way out of this dilemma, however. It is well recognized that encrypting an object is one way of allowing the classification of the object to be downgraded, so long as the corresponding decryption operation results in upgrading the classification to a value that is either equal to the classification of the key, or to the value specified in a trusted label associated with that object's contents (which must be dominated by the classification of the key). It is perhaps not so well recognized that cryptosealing an object allows the integrity of the object to be downgraded if we wish, and that verifying the cryptographic

48. The integrity domains or rules are assumed to be invariant with respect to time, although they may include the time of creation of an object as a parameter. The non-hierarchical component of the integrity marking will therefore not change. It should also be noted that downgrading the integrity of an object does not require any special privilege, unlike the case of downgrading the classification level of an object.

checksum and attached label of an object amounts to an integrity *upgrade* operation (again assuming that the integrity of the object after the upgrade is dominated by the integrity-write-limit of the cryptographic key(s) used to perform the verification).

Cryptosealing allows an object to be safely handled by low-integrity subjects, just as encryption allows an object to be safely handled by low-clearance subjects.

Therefore, as soon as a file is closed and the cryptographic checksum is written out, the file can be moved, copied, compacted, or whatever with impunity. The integrity of the resulting object will be set to null as a result of its having been written by a low-integrity subject (assuming the utility is untrusted), but the correctness of the operations will be verified and the integrity upgraded the next time the object is accessed and the checksum verified.

Because cryptosealing the object does not require or *cause* the integrity of the object to be lowered, modification, deletion or renaming of the object by a low-integrity subject would still be prohibited. This would not be a problem with a Copy, but a Move involves a Copy followed by a Delete of the previous object, so the Move program would have to be certified with respect to unauthorized deletion, *i.e.*, the copy must be completed satisfactorily before the deletion takes place. In addition, we should require that neither the Move nor the Copy program are allowed to create an output file, even a null output file, if the specified input file was not found.

Finally, we should recognize that many operating systems allow a previously existing file to be overwritten by a Move or Copy if the target file name is the same as the previous file. This may be unsafe if the Move or Copy fails. Instead, the user should either Delete the previous file before the Move/Copy, or else the Move/Copy should copy the file to the desired volume but with a temporary name so that the previous file is not destroyed. After the Move/Copy has completed successfully (checked if necessary by rereading the result), the integrity of the copied file can be upgraded, the previous file can be deleted, and the new copy renamed.

The subject that performs the Move should therefore have an integrity lower bound of 0/NULL (it can read anything) and an integrity upper bound that at least includes the integrity attribute DELETION-SENSITIVE. On the other hand, all programs that could be spoofed by the deletion of an input object (*i.e.*, those that treat File-Not-found as the equivalent of an immediate End-of-File) should have the DELETION-SENSITIVE attribute specified in their integrity-read-limit, to ensure that they do not attempt to read files that were output by programs that were not sensitive to this problem. If a File-Not-Found condition occurs when the DELETION-SENSITIVE attribute is set in the integrity-read-limit, an error will result. If a program is certified to always output an explicit null indicator or else an object with a null integrity, then it could have the DELETION-SENSITIVE attribute in its integrity-write-limit, and could either set or not set that indicator in any objects it creates, as appropriate.

But what about a program that doesn't distinguish between End-of-File and File-Not-Found, and therefore should have the DELETION-SENSITIVE attribute in its minimum integrity marking to protect itself. Does this imply that it is entitled to have the DELETION-SENSITIVE attribute in its integrity-write-limit? The answer is, "not necessarily," because the program may fail to output an explicit null when no output was intended. A poorly written Move program, for example, might encounter an immediate End-of-File on the input and fail to create an output object, yet it might still delete the input file. We therefore need to distinguish between what is required for input, and what is allowed on output.

For that reason, the terminology "minimum" and "maximum," or "lower bound" and "upper bound" as applied to the integrity markings of a subject is misleading. A better set of terms would therefore be "integrity-read-limit" and "integrity-write-limit." **In particular, there is no requirement that the integrity-write-limit dominate the integrity-read-limit.** A good example would be a linkage editor which requires valid output from a compiler as its input, even though (or rather, because) it cannot check the syntax and semantics of a source program. The linkage editor might require the attribute "GOOD-COMPILE" in its inputs, but it has no need or right to set the GOOD-COMPILE

attribute in its output. Instead, it could set the attribute GOOD-LINKEDIT in its output.

So far, we have only talked about subjects in the abstract. If a subject is assigned an integrity-read-limit, and if objects (files) are labelled with an integrity marking, then the TCB can mediate all read accesses by that subject to all objects through a high-assurance *reference monitor*, to ensure that the integrity of the object dominates the integrity-read-limit of the subject. Similarly, the reference monitor can mediate all write accesses, to ensure that the integrity-write-limit of the subject dominates the integrity marking in the label of the object.

However, subjects eventually execute objects (normally programs, but command files and source programs which are interpreted could be thought of as being "executed" as well). These objects may be classified in and of themselves, and we assume that they have been certified to have a specified integrity-write-limit or else they are untrusted. In addition, there may be constraints (integrity-read-limits) associated with the program as to the minimum integrity of any object that is read, in order to guarantee the proper operation of the program.

Because we want to be able to exchange programs and other executable objects across the network, we need to have the classification and integrity read and write limits bound directly to the object in question, rather than have the local Security Administrator set them.

We will assume that when a subject is about to execute an object, the invoking subject will first read the object (subject to the mandatory and discretionary access controls and the mandatory and discretionary integrity controls), and then create a new subject which includes that object as part of its process. The new subject will therefore inherit the following constraints on objects it can read:

- The minimum secrecy of any object which may be read is 0/NULL (unclassified)
- The minimum integrity of any object which may be read is the greater (more-dominant) of the integrity-read-limit of the object which is about to be executed, the integrity-read-limit requested by the user at logon, and the minimum-system-integrity
- The maximum secrecy of any object which may be read is the least dominant of the session classification requested by the user at logon, the clearance of the user, and the maximum-system-classification
- The maximum integrity of any object which may be read is the maximum-system-integrity.

Since the minimum secrecy and maximum integrity that can be read are only constrained by the system and not by the subject, we can define the *readclass* of the subject as the minimum integrity and maximum secrecy of objects that the subject can read.

The new subject also has constraints on objects it can write:

- The minimum secrecy of any object which may be written is the more-dominant of the minimum-system-classification[49] and the session classification level requested by the user at logon (assuming that "floating" labels and user-specified downgrading is not implemented).
- The minimum integrity of any object which may be written is 0/NULL (untrusted)
- The maximum secrecy of any object which may be written is the maximum-system-classification[50]
- The maximum integrity of any object which may be written is the least-dominant of the integrity-write-limit of

---

49. The minimum-system-classification is determined by the maximum-system-classification and the level of assurance of the TCB, in accordance with the "Yellow Books" published by NCSC, CSC-STD-003-85 and CSC-STD-004-85 and/or the level that the system is accredited for by the DAA. For example, a B2 system with a "closed" environment (cleared programmers and good configuration management controls) that has a maximum-system-classification of Secret could have a minimum-system-classification of Unclassified, while the same system, if used to process Top Secret information, would require a minimum-system-classification of Secret. Multilevel security-trusted subjects such as encryption routines and downgraders are considered part of the TCB, and can write down to the minimum-system-classification.

the program which is about to be executed (determined by the certifier), the integrity-write-limit specified by the user at logon, and the maximum-system-integrity.

Since the maximum secrecy of any object that can be written is also only constrained by the system and not by the subject, we can define the *writeclass* of the subject as consisting of the minimum secrecy and maximum integrity of objects that the subject can write.

We can therefore assign a readclass and a writeclass to an object in the integrity label, so that we will know what readclass and writeclass to assign to a subject that executes that particular object.

It should be noted in particular that the new subject is not constrained by the integrity-read-limit of the parent subject. The "integrity" of the object that is about to be executed is the integrity-write-limit that it was certified for. If a subject is trusted to obey the rules of its integrity domain, then it is trusted to create an object that has that same integrity domain, and the hierarchical integrity value associated with the object is the probability that the creating process created the object correctly.

It doesn't matter to the invoking subject how a given program was written — only that it was certified as having a certain integrity. If the program is capable of evaluating all possible inputs (*e.g.*, a UDI), then that is well and good. On the other hand, if the program relies on the TCB to protect itself from untrusted input by means of a specific integrity-read-limit, and it was certified subject to that assumption, then that is also acceptable.

Ideally, we might like to restrict a subject's read access or integrity-read-limit (level and categories) on a file by file basis. Just because a TP is prepared to accept and screen untrusted input (a UDI) coming from the user's terminal, for example, does not mean that it can validate all of an existing database before it updates a record. Similarly, a linkage editor may accept untrusted input from the terminal that directs it to include certain object modules in its input and to use a certain link library, yet the object modules and link library itself may have a higher level of integrity

In the case of a subject that reads a file, this approach might be practical. Most programs access a file by means of a file descriptor, "handle," or DDNAME, so that the particular file name that is to be accessed can be specified at run time as a parameter. The integrity-read-limit could be associated with the file descriptor as part of the program, along with the other attributes of the file. The TCB could then enforce the integrity-read-limit when the file is opened, returning an error if the file does not meet the integrity requirements. Another alternative would be for the TCB to return the integrity of the file at open time, and let the program decide whether that was sufficient. If the program has been certified, either approach would be adequate, and would provide considerable flexibility.

However, other objects, including simple memory objects such as parameters, do not have as rich a structure as a file. With the exception of some object-oriented operating systems, subjects generally do not access such objects through a type-descriptor that could specify the integrity-read-limit. In addition, one of the objectives of building a reference monitor is to minimize the amount of code that has to be trusted, and to provide a containment mechanism that will assure the correct operation of untrusted programs.

For that reason, it suggested that the integrity-read-limit associated with a subject be applied to all objects read by that subject, by default, but that the program be able to override the default in the case of files. This integrity-read-limit can be defined in the readclass of the object at the time it is certified, and will apply to all objects that are read for which a specific integrity-read-limit has not been provided by the program. If a program or other object has not been certified as a TP, then the integrity-read-limit and the integrity-write-limit are both 0/NULL.

These controls should be implemented on a high-assurance TCB that includes a reference monitor. At a minimum, the system architecture

50. Some subjects may write objects which they cannot later read, *e.g.*, the audit log. Processes may also have to signal (an interrupt, followed by a read down) to higher classification-level processes for synchronization purposes.

requirements TCB imposed by the TCSEC for a B2 TCB are considered necessary for the protection of objects from viruses running at the same privilege level as the user, and higher levels are desirable. The features in bold type are particularly critical:

"The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (*e.g.*, by modification of its code or data structures). **The TCB shall maintain process isolation through the provision of distinct address spaces under its control.** The TCB shall be internally structured into well-defined largely independent modules. **It shall make effective use of available hardware to separate those elements that are protection-critical from those that are not.** The TCB modules shall be designed such that the principle of least privilege is enforced. **Features in hardware, such as segmentation, shall be used to support logically distinct storage objects with separate attributes (namely: readable, writable).** The user interface to the TCB shall be completely defined and all elements of the TCB identified."

When it comes to implementing these concepts within a TCB, there is a subtlety with respect to passing parameters to multilevel integrity-trusted subjects that might be overlooked. Low-integrity subjects may invoke high-integrity subjects, and the high-integrity subjects may read or copy low-integrity objects (parameters) from the low-integrity subject, if and only if the high-integrity subject is a multilevel integrity-trusted subject, and if the integrity of the parameter being read is greater than the integrity-read-limit of the high-integrity subject. The high-integrity subject may return a result by having the low-integrity subject read up, or else the high-integrity subject could directly store the result in the low-integrity subject's memory, assuming the low-integrity subject passed the address of a variable in its memory.

However, in order to protect against a virus or Trojan horse program altering the contents of memory, a low-integrity subject cannot be allowed to write into the memory of a high-integrity subject. As a result, if a high-integrity subject calls a lower-integrity subject for some purpose, the passing of parameters can be in one direction only. The low-integrity subject may *not* store a result back in the high-integrity's memory, so for example the high-integrity program cannot provide a read buffer for a low-integrity program to use while reading a disk. Instead, the high-integrity program will either have to create a low-integrity buffer for this purpose, or the low-integrity program will have to provide a buffer which the high-integrity program can read.

In summary, trusted multilevel-subjects can read down to their integrity-read-limit, assuming that they are certified to handle the untrusted information properly. But no subject can be allowed to write up beyond its integrity-write-limit, not only because to do might mislead the high-integrity subject, but because a virus or Trojan horse program might alter the programs used by the high-integrity subject.

4.5 Dynamic Mapping of Integrity Categories

In order to be useful for commercial purposes, integrity categories should be "plentiful" and easy to create. Unlike security categories or compartments, which are relatively few in number, usually have very explicit meanings, and are usually coordinated on a centralized basis, integrity categories may be used to provide an assured pipeline between two programs, so that data is guaranteed to be produced by one trusted process and read or updated by another. The intermediate data file may only be written and read by those two programs, and never referenced by any others. Nonetheless, it is desirable that those two programs could create a unique integrity category, shared only between those two programs for one instance of the intermediate output and used strictly for that purpose, without having to involve a central coordinating authority.

As Karger[51] correctly observed, "The major difficulty with using integrity categories to protect CDIs and TPs seems to be one of management. A large system, like IBM's AAS, may have thousands of distinct TPs, while most lattice model designs to date have considered 64 categories to be a large number! Furthermore, categories tend to be managed centrally by a

51. Karger, Paul, "Implementing Commercial Data Integrity with Secure Capabilities," **Proceedings of the 1988 IEEE Symposium on Security and Privacy**, April 18-21, 1988, Oakland, CA: IEEE Computer Society Press, Washington, DC, 1988, pp 130-139.

single security authority, while the security policy in [a] commercial transaction system probably needs to be more decentralized, with individual applications managers creating TPs and granting authorizations." In a very large network, it would obviously be almost impossible to coordinate the names of all of the various integrity compartments that might be desired across the entire network, for this is the problem that led us to reject the global Access Control List approach.

Without attempting to constrain possible implementations, it is apparent that an approach that assigns a particular bit in a fixed-length field or label to permanently indicate a particular integrity category will probably not be adequate, for the size of the field would tend to grow without bounds in a large network. In addition, there is the difficulty of coordinating a particular integrity category name with all of the other potential users of integrity categories, in order to avoid accidentally reusing an integrity category with unintended consequences. Finally, some categories of information dealing with financial records, contracts, and other matters of lasting import must be protected for an arbitrarily long amount of time, so integrity categories will tend to be "eternal," yet the demand for new applications will require that new categories be invented from time to time.

For this reason, it is suggested that the "name" of the integrity category could be a randomly or pseudo-randomly generated *passphrase* with 2^{128} (3.4×10^{38}) or more different possibilities, so that there would be an extremely low probability that any other program or process would have accidentally picked the same name (assuming that the passphrase is kept secret). Assuming that the average user's working vocabulary contains approximately 20,000 words, and that 10,000 or so are in the convenient range of 3 to 8 characters in length, randomly selecting 10 passwords to make up a passphrase would provide the necessary variability. According to the Birthday Problem in statistics, approximately 10^{19} randomly chosen passphrases would have to be selected in order to have a 50% chance that two passphrases would accidentally be the same.

Assuming such a lengthy integrity category name is selected, a cryptographic checksum technique should be used to reduce the passphrase to a 128-bit label for greater economy in storage and transmission. We clearly don't want to have a label that includes 2^{128} bits, one for every possible category!

Instead, we should dynamically *map* that huge name space to a much smaller number of bits as integrity-labelled data is accessed and deaccessed, for the sake of efficiency. This is precisely what compilers do when looking up names in a name table — a hash table lookup process is invoked against the already existing names, the name itself is checked against any previously stored names that have been assigned to that slot in the table, and if a collision occurs the new name is moved to some other position. This name lookup process only has to occur when a new integrity category is encountered — after that, an appropriate bit can be assigned in a standard label.

The only issue is how many bits have to be provided in the label, and that is a function of how many integrity categories have to be simultaneously accessed at any given point in time. That in turn depends on how a category is opened or accessed, and more importantly on how a category is deaccessed. One possible mechanism would be for the TCB to increment a use count associated with the integrity category each time a subject with that integrity category is created within the TCB, and to decrement the use count when a subject with that integrity category is deleted. In particular, if files are normally stored in encrypted form and have to be decrypted when they are read into the TCB, then the mapping from the 128-bit integrity category "name" can be performed at that time. When the user who was using that category logs off, all unencrypted information will be destroyed, so the use count can be deleted at that time.

When the use count reaches 0, that category is no longer in use by any subject, so the entry can be removed from the hash table (this is a little tricky to implement) and the corresponding bit in the access control label could then be reused. However, if an object previously imported into the TCB by a subject was still in memory, reusing the bit in the access control label could cause possibly allow another subject to access one of those objects by mistake, so it may also be necessary to keep a use count of all objects created with a particular integrity category. Only when all of the subjects and all of the objects associated with a given category have been

deleted from the TCB would it be safe to reallocate the label bits.

It may seem that this solution is somewhat extreme, in that it sometimes requires deleting objects from the TCB that we would like to protect! But what we are really talking about is how to most efficiently manage subjects and objects within the main memory of the TCB, and how to check the dominance relationship each time a subject accesses an object. We may therefore choose to restrict the implementation of this dynamic mapping technique to only those subjects and objects that are stored within main memory — access to files stored on external media could compare the 128-bit integrity category name rather than the dynamically bit-mapped object label used for memory objects. The overhead of doing the name table lookup would only occur when the file is opened, and in most cases the computation time will be modest compared to the time required to access the first record.

If two users privately agree to establish an integrity category using a secret integrity passphrase, then only they will be able to create or reference objects with that category. By convention, one or more of the words in the passphrase could be changed to include the date or a generation or version number in both the subject and the object, so that previous instances of a file could not be confused with the current version. This passphrase technique will be called an *informal* integrity category, because the particular category does not have any formally-defined or centrally-controlled meaning but is simply being used to enforce an assured pipeline.

On the other hand, there is also an obvious requirement for *formal* integrity categories with very specific meanings that are controlled through a centralized registry approved or controlled by the DAA for that community-of-interest. For example, we would hope that a nuclear reactor control program would not be placed into actual use until its design had been carefully reviewed by human experts from the Nuclear Regulatory Commission, and the implementation tested and certified by a duly authorized individual or individuals after an extensive scrutiny of the design, including simulation testing. Only those authorized experts would be allowed to assign the "nuclear reactor control" category to an object, or to upgrade the hierarchical integrity level of the program from that required for "test" to a value suitable for "production."

Because there are a number of independent national and international agencies who might have an interest in formally certifying the integrity of subjects or objects, it is suggested that up to 255 countries or supranational organizations (*e.g.*, NATO) be allowed, with 255 national agency codes per country. Up to 65535 different formal integrity categories could then be made available to be allocated by and for each national agency. Within each category, an arbitrary number of subcategories can be provided through the list technique, with subjects being granted or denied the ability to create or modify information with any or all of these categories and subcategories. Even though formal integrity categories (like secrecy categories) can never be reused and must therefore be eternal, that number of categories should surely suffice for the foreseeable future. The DAA would have to make sure that there is agreement between countries and agencies as to the meaning of a country code, national agency code, and integrity category, with a mapping being defined between the different agency's numbering schemes if necessary. (A "foreign" agency cannot spoof a user by creating a false integrity category, because without the approval of the DAA the digital signature of the originating TCB will not be recognized as valid by the receiving TCB.)

In addition to this mechanism, integrity subcategories could be implemented if required by having each bit in the label implicitly refer to the beginning of a list of subcategory names. Presumably subcategories will not be needed very often, and not very many would ever be created, so the time to search the list would be short. If only a small number of categories is anticipated, the list approach could be used for the categories instead of the hash table approach, just as was done for the external files.

The total amount of information required for the compressed integrity category names in the object label is therefore 128 bits per informal integrity category, or 32 bits for each formal integrity category with no subcategories, which is not an unreasonable amount of information to carry in a file label. Assuming that not more than 40 to 50 integrity categories will ever be opened

simultaneously within one TCB, these categories can easily be mapped to a 64-bit access control field for use by a lattice model.

In those instances where a static separation of duties is required, the individuals involved can be separated into two sets, with one set of individuals being authorized to use one integrity category and the other set being authorized to use the other. Either formal or informal integrity categories could be used. The TP could then be certified to require two inputs, one with one integrity category and the other with the other one, and the separation of duty requirement would then be satisfied. The DAA or other approval authority for the particular application would be responsible for keeping the two sets of individuals separated, and the DAA would have to ensure that the approval authority for a particular program was not granted the unilateral ability to execute a program he approved.

Such a mechanism still requires that someone keep a list of what individuals are authorized to do what, but it allows the list to be decentralized to the individual application and does not require global coordination across multiple hosts, since it is only the host where the process is executed that has to enforce the separation-of-duty requirements. On the other hand, if multiple hosts are required, the individuals in question can carry their own digitally-signed certificates to the hosts in question, thereby avoiding the problem of coordinating a global database of authorized users for specific applications across the entire network.

4.6 Integrity Attributes

In addition to the integrity categories used to represent application-dependent integrity domains, a small number of predefined categories may be considered global integrity attributes or options.

The globally-defined integrity attributes should include the following:

VERIFY: If VERIFY is specified in the integrity-read-limit of the subject, then in the case of a read or modify operation all other operations by the subject shall be suspended, the object read in its entirety by the TCB, and the cryptographic checksum recomputed for the entire object and compared to the checksum in the integrity label, *prior* to the subject taking any effective action based on any of the information contained within the object. If the checksum does not verify the integrity of the object being read is defined to be 0/NULL, and either all further processing shall be aborted, or the integrity of all further outputs shall be set to 0/NULL. If VERIFY is specified in the integrity-write-limit of the subject, then in order for the VERIFY attribute to be set on the output object in the case of a write or modify operation, when the object is closed the object shall be reread by the TCB from beginning to end and the checksum recalculated based on the data actually recorded, and compared to the previously calculated and recorded checksum to detect any I/O errors while the object was being written out. If the checksum does not compare, the TCB shall write or rewrite the integrity label of the object to indicate that an error has occurred. (The checksum can be included in an integrity trailer label placed at the end of sequential files or telecommunications sessions which cannot conveniently be overwritten, in order to ensure that the correct checksum and integrity marking is written out.) The intent of VERIFY is to assure that applications which are not prepared to back out the changes that might be necessitated by their reading potentially erroneous data (a fact that might not be discovered a considerable amount of data had been processed) are not allowed to access such data until its integrity has been verified. Because of the additional overhead, this option should be used judiciously.

RECORD: The RECORD attribute implies that the file contains records that have been individually authenticated, so as

to permit nonsequential I/O operations without (re)reading the entire file to calculate or record a checksum. In order to detect any addition, deletion, replication, or reordering of records in a RECORD file, each record must contain a logical record number which is included in the checksum. The logical record number must agree with the physical record number after adjusting for the location of the start of the file on the volume, in order to ensure that records cannot be inserted, deleted, or reordered. The total number of records in the file must be recorded in the file label, and checked when the file is first opened (for read) or closed (for write) to ensure that no records are missing from the end of the file. RECORD and VERIFY are independent of each other, except that subjects performing non-sequential I/O may be required to specify either RECORD or VERIFY in their integrity-read-limit in order to ensure that all data are properly authenticated during read operations. Programs which authenticate the individual output records may set the RECORD attribute in the object created or updated.

PEDIGREE:

This option requires (in the case of the integrity-read-limit) or provides (in the case of the integrity-write-limit) the collection of auditing data in any objects created or modified by a subject, so that the flow of information that led to the creation or modification of such an object can be clearly identified. This auditing data includes the name or other identification of the process used to create the data; the date, time, and place of execution; the name and unique ID of the user accountable for the process; and the object name and sufficient other information to uniquely identify the contents of all of the objects that were input to that process to create the result, in accordance with Rules 10 and 11.

DELETION-SENSITIVE:

This option requires (in the case of the integrity-read-limit) or provides (in the case of the integrity-write-limit) that the subject differentiate between an End-of-File and a File-Not-Found condition, so that the accidental or deliberate deletion of an object can be detected. If the subject has the DELETION-SENSITIVE attribute and it attempts to read a object that cannot be found, the process will be aborted or all further objects created by that subject will be set to a null integrity by the TCB. If the DELETION-SENSITIVE attribute is specified in the integrity-write-limit, then the subject is trusted to always provide an explicit End-Of-File indication if an only if a null output is intended, rather than simply failing to output an object in that case.

4.7 Integrity Covert Channels

Because of the duality between the Bell-LaPadula security policy model for sensitive data and the Biba integrity policy model for critical data, it is necessary to consider whether integrity covert channels exist, what they would mean, and how they could be controlled.

The TCSEC defines a covert channel as "any communications channel that can be exploited by a process to transfer information in a manner that violates the system's security policy. There are two types of covert channels: storage channels and timing channels. Covert storage channels include all vehicles that would allow the direct or indirect writing of a storage location by one process and the direct or indirect reading of it by another. Covert timing channels include all vehicles that would allow one process to signal information to another process by modulating its own use of system resources in such a way that the change in response time observed by the second process would provide information."[52]

From the standpoint of that definition, it is obvious that integrity covert channels could exist — whatever covert channels exist in a system that could violate the Bell-LaPadula policy could potentially be used to violate the Biba integrity policy, for example.

Furthermore, if an integrity covert channel were to exist, it could potentially be much more harmful than a disclosure covert channel, because of the difference in sensitivity to the bandwidth of the covert channel. Although it is conceivable that a single bit of information could have devastating consequences if it were disclosed (*e.g.*, "The Allied invasion of Normandy will/will not take place via the English Channel."), in most circumstances an appreciable amount of information must be disclosed before considerable damage would result. For that reason, the bandwidth of the covert channel must be considered. Because covert storage channels often arise in conjunction with process synchronization and other operations that are intrinsic to the operating system and would cause considerable performance degradation if they had to be removed entirely, the TCSEC states that covert channels of less than 1 bit per second are generally considered acceptable in most application environments, particularly if covert channels whose bandwidth potentially exceeds 1 bit in 10 seconds are subject to audit.[53]

In the integrity case, however, a covert channel could potentially be very harmful even if only a *single* bit were transmitted, since that bit might serve as an external signal to a Trojan horse program to carry out whatever damage function it was to perform. In that sense a single bit might be the equivalent of the famous "Climb Mount Niitaka" message that was sent in an open code to the Japanese forces preparing to attack Pearl Harbor, directing them to proceed as planned.[54]

Since it is presumably possible for a low integrity subject to send a signal to a high integrity subject through a covert channel, we have to consider what the implications would be. In the case of a conventional secrecy-type covert channel, an untrusted Trojan horse program which is executing on behalf of a cleared user and processing classified data somehow manages to signal that information to a confederate (a subject with a lower clearance) through a covert channel, and the dangers are obvious.

In the case of an integrity covert channel, however, we have the situation where the low-integrity untrusted[55] process is attempting to signal to a high-integrity *trusted* process. The obvious question is, "So what? If the process that is being signaled to is in fact trusted, shouldn't it just ignore or reject the untrusted input?"

In considering this question, we have to think more carefully about the reason for the integrity confinement property (integrity *-property) that prevents an untrusted process from writing up in integrity and attempting to spoof a more trusted process. If the receiving process is trusted, why is the confinement property necessary?

The answer has to do with the Sorcerer's Apprentice problem — a process may be trusted to do exactly what it is told to do, and only that, but it may still be "dumb" or **unsafe**, *i.e.*, incapable of rejecting certain inputs as inherently undesirable. For example, we might hope that the missile-targeting program for an strategic

52. A brief discussion of the problem of covert channels in networks, together with a good bibliography, is contained in Eggers, K. W. and P. W. Mallett, "Characterizing Network Covert Storage Channels," **Fourth Aerospace Computer Security Applications Conference**, Orlando, FL, December 12-16, 1988, Computer Society Press of the IEEE, Washington, DC, 1988, pp 275-279.

53. It must be noted that these guidelines would still allow more than 1000 bytes of information per day to be signaled over a covert channel. Hamming codes and other compression techniques could effectively double or triple that figure, depending on the redundancy in the text. Forward Error Correction techniques could be used to eliminate the effect of a noisy channel, if necessary. In particular, covert timing channels between cooperating subjects in a multitasking environment would seem to be almost impossible to eliminate unless an extremely rigid task scheduling discipline is used so that neither any variations in a subject's CPU utilization *nor the use of I/O resources* can be communicated to another subject. Tasks must be given a fixed amount of time every time they are dispatched, whether they can use it effectively or not. Likewise, a WAIT for I/O to complete must take the same amount of time (sufficient to complete the read or write operation with a high probability) in every instance, regardless of when the I/O operation actually completes. A similar technique in the communications field is the use of full-period link encryption to deny the enemy any knowledge of traffic patterns or addresses. It remains to be seen whether it is possible to build systems that are cost-effective under these constraints. Hopefully, the continuing increase in performance and decreasing price of personal computers and dedicated workstations will make issues of efficiency and performance less important than in the past.

54. Kahn, David, **The Codebreakers**, Macmillan Publishing Co., New York, 1967, p. 41.

55. For the purposes of this discussion, untrusted/trusted is used in the integrity or "correctness" sense, not necessarily having anything to do with secrecy.

ICBM would check to ensure that the coordinates of the target that were specified were not within the boundaries of the country that launched it. However, a tactical missile might not be subject to such constraints, since that might limit its defensive use in the event of an invasion of the homeland by the enemy. In the case of the tactical missile it would therefore be quite important that only trusted data be processed by the trusted but unsafe subject.

For this reason, it is necessary to enforce the default integrity-read-limit on all objects that a higher-integrity process might access. In addition to applying to files, this constraint will also apply to signaling mechanisms. A lower-integrity subject will be allowed to signal or call a higher-integrity process if and only if the integrity-write-limit of the invoking process dominates the integrity-read-limit of the invoked process. Otherwise, even the fact of the interrupt, much less any information passed along with the interrupt such as the reason, might convey low-integrity information that the high-integrity process was not certified to handle.

However, reading data from the normal input medium is one thing, but for a trusted program to read and act upon data from a covert channel mechanism would be quite something else. Normally there would be no reason to suspect that a trusted program would ever act upon such illicit information, even if the data conformed to all of the syntactical and semantic rules of legal input as though it had been read via the normal channels.

Unfortunately, the difficulty once again is that of trying to prove a negative proposition, *i.e.*, that a program does what it is supposed to, *and nothing else*. It might be fairly simple to examine the source code of a cosine routine and conclude that it would never modify a file or even the input argument, *i.e.*, it is a pure function and has no side effects. However, it would be quite a different matter to prove that a word processing program or DBMS program that routinely reads and writes files would never modify any file other than the user was currently working on. It would greatly simplify matters if the TCB could provide a containment mechanism so that we wouldn't have to worry about such things.

The Mandatory Integrity Controls as previously defined provide a considerable amount of assistance in this area, especially the requirement that a subject cannot modify, delete, or rename a object unless the integrity-read-limit of the subject dominates the integrity of the affected object (*i.e.*, the subject can legitimately read the previous object), *and* the integrity-write-limit of the subject dominates the integrity of the affected object (*i.e.*, untrusted processes cannot destroy more trusted data), *and* the integrity marking of the modified or replaced object is dominated by the integrity-write-limit of the subject.

Because a low-integrity subject cannot modify, delete, or rename a higher-integrity object, and since the integrity markings include the integrity categories that correspond to the integrity domain(s) that a given process was certified against, it follows that the most obvious form of what some people (but not all) would consider a denial-of-service attack, namely the accidental or deliberate destruction of data by an unauthorized subject, will be prevented by the TCB through the application of the Mandatory Integrity Controls. In order for a subject (program or human user) to be able to create, modify, delete, or rename an object, the subject must be certified as trusted for all of the integrity categories to the level required by the hierarchical integrity component of the integrity marking of the object.

The obvious application of the Mandatory Integrity Control policy to external devices and actions would be to treat command channels (for example, the ignition control signal which launches a missile) or other I/O devices or actuators as either subjects which can only read objects whose integrity marking dominates their integrity-read-limit, or alternatively as existing objects whose contents can only be modified by a subject whose integrity-write-limit dominates the integrity marking of the object.

However, we previously assumed that if a process was not explicitly designed, constructed, tested, and certified as conforming to a set of syntactic and semantic rules that make up an integrity domain, that process must be considered untrusted with respect to that domain. For that reason, unless the rules that make up an integrity domain are very carefully specified to ensure that a process does exactly what it is supposed to do and nothing else, it appears that we cannot conclude that any process is necessarily safe with

respect to an integrity domain that it was not specifically designed to handle properly, regardless of whether the inputs were provided by an overt or covert channel.

However, we can conclude that the Mandatory Integrity Controls if implemented properly will prevent any modification, deletion, or renaming of any object by a subject of lower integrity, or the initiation or termination of any external action under the control of the TCB by a subject within the TCB which was not certified as having the required integrity markings.[56]

There is still one significant area where an integrity covert channel might be a problem, and that is the case of an accidental or deliberate denial-of-service situation that might result from an integrity covert channel being used to signal from a low-integrity subject to a higher-integrity subject, causing the higher-integrity process (which was not designed, constructed, tested, or certified *not* to take such actions, perhaps because no one ever thought of the possibility) to monopolize scarce resources or cause a deadlock in such a manner as to deny a specified service to a group of otherwise-authorized users for a period of time that exceeds the intended and advertised maximum waiting time for the service.

As defined by Yu and Gligor[57], "Denial of service can be both a safety and a liveness problem. It takes place whenever one or both of the following situations occur:

- Some users prevent some other users from making progress within the service for an arbitrarily long time.
- Some users make some other users receive incorrect service, *i.e.,* the service does not satisfy its intended specifications for the latter users."

Instances of the first case are considered liveness problems and generally involve the excessive consumption of resources, while instances of the second case are considered safety issues and include situations such as a permanent deadlock between user processes. (Note that Yu and Gligor's use of the term "safety" is slightly at odds with the previous use of the term "unsafe" in conjunction with the missile launch program.)

As an example of how such a situation might arise, suppose that a subject such as the I/O scheduler of the operating system has been certified to read and write data with a very high degree of integrity. Such a task might run at a very high priority in the system, and there might not be any other robustness mechanism in the operating system that would ensure that a single task cannot dominate the system. Although such a program might read and write data flawlessly for years, an integrity covert channel could conceivably be used to signal the otherwise trusted program and suddenly cause it to consume all of the available processor time, memory, disk space, or whatever — a liveness problem. Although such a flaw might be relatively apparent during a design or code review, deliberately creating a deadlock between two cooperating Trojan horse programs would be much harder to detect because of the difficulty of analyzing asynchronous, interrupt-driven processes.

Unfortunately, although secrecy restrictions may sometimes be ignored during a crisis (for example during battle to get intelligence information to the field as quickly as possible), integrity and denial of service are generally of lesser concern *except* during a crisis, when they may be fatal. The possibility of an integrity covert channel leading to a denial of service attack therefore has to be taken quite seriously, especially in a networking context.[58]

Yu and Gligor's paper appears to be very promising from the standpoint of providing a formal specification and verification method

---

56. Saying so in English is not the equivalent of proving these assertions mathematically, of course. I look forward to someone reducing the proposed Mandatory Integrity Control policy to a formal model, and then proving that these assertions follow from the model.

57. Yu, Che-Fn and Virgil Gligor, "A Formal Specification and Verification Method for the Prevention of Denial of Service," **Proceedings of the 1988 IEEE Symposium on Security and Privacy**, April 18-21, 1988, IEEE Computer Society Press, Washington, D.C., 1988, pp 187-202.

58. In this connection, it should be observed that the "Yellow Books," CSC-STD-003-85 and CSC-STD-004-85, "Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments" issued by the National Computer Security Center do not address the possibility of either integrity threats or denial of service, but only the threat of compromise when coming up with the security index matrix (data sensitivity vs. minimum user clearance) which

which features the use of service and user agreement specifications as a means of addressing the denial of service problem. It appears that user agreements can be codified to include a specification of what constitutes safe-service and live-service invocations, together with a formal service specification, using a temporal-logic-based specification language to deal with the issues of concurrency, scheduling, fairness, etc. If so, and assuming that these specifications can be used in a practical way to evaluate cooperating processes, then it would appear that the denial-of-service problem can be converted into an integrity problem, and that would be good news indeed.

4.8 Discretionary Integrity Policies

In most cases, the proper rejoinder to the statement "Object X has hierarchical integrity N with respect to integrity domain A" should be, "Says who?"

This question is analogous to the Discretionary Access Control requirement of the TCSEC, which allows a user to exercise his discretion as to whether to reveal information to someone who has the appropriate clearance, based on that user's assessment of the recipient's Need-To-Know. In the integrity case we are asking what individual in combination with what trusted process created that object, and by implication assigned that object a specified integrity rating, because we intend to make a decision as to whether or not to accept, believe, obey, or trust that object based on that individual's authority and believability.

Whereas the Discretionary Access Control policy deals with Need-To-Know, the Discretionary Integrity Control policy deals with *Need-To-Do*.

Discretionary Integrity Controls are identity-based access controls that operate at the discretion of the receiving subject, as opposed to integrity category controls which are imposed on a mandatory basis by the central accrediting authority and enforced by the TCB in order to protect the subject. That does not denigrate the use of discretionary controls — it just means that the subject must take on more of the responsibility in this area. In particular, before a particular program is accredited or certified as trusted with respect to some integrity domain, the use of discretionary integrity controls may have to be examined in considerable detail. On the other hand, the human end-user may be free to accept or reject a particular object based on the identity of the originator, as he or she sees fit.

In particular, certain individuals may be rejected as untrustworthy or subject to a potential conflict of interest. To repeat a point made much earlier, the fact that Discretionary Integrity Controls are applied when the data is read means that if the originator of some object is discredited after the fact, that object can be easily be rejected even though it had been accepted previously. In the intelligence community, for example, it was revealed during the "Irangate" hearings that the CIA and the White House staff had continued dealings with a Mr. Ghorbanifar, even though he had failed a polygraph examination. A less sensational example would be a programmer responsible for the credit/debit program for a bank. A prudent bank would probably not allow that programmer to run that program on live data without other checks and balances, for obvious reasons.

Similarly, a programmer should not be allowed to accredit that program, and likewise someone who accredits such a program should not be allowed to run it, because of the possibility that either could potentially take advantage of some flaw. For that reason, **more than one individual may be required to have approved the contents of the data object in order to enforce the requirements for separation of duty.**

A combination of mandatory and discretionary integrity mechanisms can provide the equivalent function as the triple of (UserID, TP, and [CDI_1 ... CDI_n]) proposed by Clark and Wilson in their rule E2 to control who can exercise a given TP against a given set of CDIs:

- Mandatory and discretionary *access* controls can control what users are allowed to execute a process (TP).
- Mandatory and discretionary *access* control categories can also be used to control which subjects (processes and users) can access particular input CDIs.

---

specifies what level of assurance is required for a system processing classified or sensitive information. From that standpoint the guidelines are not very conservative, and should be regarded as the minimum requirements.

- Mandatory *integrity* control categories associated with the subject process can limit what input CDIs a given TP is allowed to accept or believe, and the same or other integrity categories can be used to mark what output CDIs are created to provide an assured pipeline. The separation of duty requirement can be satisfied on a static basis through the use of two different formal integrity categories.

- Discretionary *integrity* controls can make use of an unforgeable mechanism to indicate which user executed the process that created a certain CDI, so that some later TP will be able to confirm that the CDI in question was produced by an authorized user, and accept or reject the object appropriately.

This mechanism is considerably more flexible than the database-of-triples mechanism proposed by Clark and Wilson, in that the TP itself can access the discretionary integrity control mechanism to determine whether the input data is to be considered valid.

For example, suppose that a company's policy is that the signature of the requester's Manager must be obtained for all purchase requests, that in addition the signature of the requester's Director must be obtained for purchases over $1000, that a Vice President's signature is needed for all purchases over $10,000, that the signature of the Contract Administrator assigned to the particular contract is required for any purchases charged to a Government contract, that the signature of the Comptroller is needed for all indirect capital acquisitions, and the President's signature is required for all expenditures over $100,000. Furthermore, the Contract Administrator has delegated his signature authority to the Deputy Contract Administrator for contracts A, B, and C, and an offsite Operations Manager can sign for his Vice President for requests originating within his organization, up to $5,000.

Such a separation of duty policy would not be particularly unusual, but it would be dynamic rather than static since it takes into consideration the value of the data items themselves. The complexity of the interrelationships of reporting organizations, contract amounts, and dollar values of purchases would undoubtedly require access to an application database, which would in turn have to be managed by other programs.

The point is that if a general purpose TCB were to attempt to provide this level of flexibility in controlling which users can exercise what programs and access specified input and output CDIs, the complexity of the TCB would very quickly grow to the point that the system could not be formally evaluated. And if the objects (CDIs) representing purchase requests were collected and transmitted over a network instead of residing at a single monolithic TCB, thereby requiring the global administration of such a policy, the task of ensuring the integrity of the results would probably become impossible.

In contrast, if the TCB were to provide a mechanism whereby the user who exercised the TP and/or approved the output was irrefutably identified in an integrity label associated with each output CDI, then a certified, trusted application program could assure itself that the appropriate levels of signatures were provided, and the operating system itself would not have to be involved in maintaining that policy.

It should also be pointed out that such a mechanism makes it easy to deal with objects *that have not yet been created*, whereas the rule-based mechanism proposed by Clark and Wilson is obviously constrained to deal with already existing objects.

However, the proposed system does not provide quite the degree of granularity suggested by Clark and Wilson without one modification. In the lattice model suggested above, if a user has access to the TP and also has access to a given set of CDIs, and if the TP can access those CDIs on behalf of any user, then the access would be allowed. In contrast, the use of a true triple consisting of the user, TP, and a set of CDIs would make it possible to specify exactly which CDIs a given TP can access on behalf of a particular user. In other words, in the lattice model a user who had valid access to a TP and to a set of CDIs could input any CDI into that TP that the TP would allow for any user, whereas the Clark and Wilson model would allow constraining a user to inputting only specified CDIs into that TP.

An example of such a situation would be a clerk in a Personnel Department who is allowed to input salary data into the payroll database. Each clerk could invoke the TP which updates the payroll database, and the payroll database would accept inputs pertaining to any employee in the company, but a clerk should not be allowed to input a record which pertains to his or her own salary level.

Assuming that this application is representative of those cases where the flexibility of specifying CDIs that a TP can access on behalf of a specific user is required, then the functionality of Clark and Wilson's triple can be provided by requiring the discretionary integrity control function to include the capability of *excluding* an individual or group of individuals, in a manner that is similar to the exclusionary discretionary *access* control requirement for Class B3. This would allow the payroll clerk to read his or her own payroll record (assuming that the Discretionary Access Control policy did not prohibit it), but the TP would reject any attempt by a clerk to change his or her own payroll record.

The dynamic separation-of-duty requirement will normally be rather somewhat complex, since in most applications it will deal with records and fields rather than files, and the granularity of the TCB may not extend to what amounts to nested objects within objects.

For that reason, it is suggested that the TCB should *not* be responsible for enforcing the dynamic separation of duty requirements imposed by a TP.

Instead, the TCB should assure that the originator of an object is accurately identified and provide a mechanism whereby the trusted (certified) application program or TP can know the invoking user as well as the originator of particular objects and can thereby implement whatever separation-of-duty policy is required by the human accreditor.

Because the TP is responsible for enforcing the dynamic separation of duty requirements, Discretionary Integrity in this case is at the discretion of the program developer and accreditor. The human user can decide for himself whether he wishes to *reject* some information based on the identity and authority of the originator, but in the case of separation of duty the human user cannot override the TP and decide to *accept* something that the TP decides is not acceptable.

4.9 Digital Signatures

It is obviously necessary to prevent *anyone*, including both the originator and the recipient of an object (who may know the cryptographic key used to encrypt the object) from being able to change the access or integrity control labels of that object (or contents of the object) without detection. For that reason, the TCB must digitally sign both the access and integrity control labels, which include the name and cryptonym of the authorized user who initiated the TP which created a given object. In this respect the TCB is acting on behalf of the Designated Approving Authority (DAA) or system administrator to enforce the immutability rule, Rule 1, and the attribution rule, Rule 2.

In many cases, the accurate identification of the originator will be sufficient to ensure that sufficient integrity is provided. However, in the case of critical data or processes, it is frequently necessary to require that a higher standard of care be exercised, and to impose some liability for *malfeasance* by the originator.

But the fact that the user created an object does not necessarily mean that he or she approved the contents — the object could have been a rough draft, a trial balance, a simulation or test, or simply be incorrect, and this is not sufficient to prove the *intent* necessary to establish malfeasance. It is therefore necessary for the subject to be able to require a non-repudiation mechanism which reflects the usual implication of a written signature in commerce:

A user's digital signature must be of his own volition; and must express his conscious, knowing, and willful agreement with the contents of the signed object.

For that reason, the user must be provided the opportunity to review the output while the object is being created and is still held securely within the TCB, and then to add his digital signature and/or comments to the provenance of that object if he approves. In addition, a user should be able to review, comment, and digitally sign the provenance after the object has been created.

However, in either case the process that is used to present the data to the user for his review and signature must be trusted to display *all* of the information in the file to the user, and not to modify the information before the digital signature is affixed. Finally, the TCB must be trusted not to retain and possibly misuse the user's private signature key, just as it must not retain a user's private decryption key.

The fact that an object bears a digital signature does not necessarily mean that it is trustworthy, however, for even the best digital signature scheme is worthless if the individual signing the document is a pathological liar. Therefore, part of the discretionary integrity controls associated with a subject or TP must provide either the TP or the human user the ability to decide or specify exactly whose digital signatures must be present in the integrity control label associated with an object, and in what combinations, in order for that object to be believed or not to be believed.

In addition, multiple signatures provide a means of enforcing separation of duty (approval signatures) that is in addition to integrity categories. A multiple signature capability would also support contract signatures between multiple parties, and provide a notarization or witness function. Although the TCB provides protection against an attempt by one of the parties to renege on a signature,[59] an additional human witness may be desirable to attest to the apparent soundness of mind and absence of any compulsion over the signer. It should be noted that someone can witness or notarize a document that someone else has signed, without necessarily understanding or even reading it. For that reason, and to differentiate between those users who sign a document indicating their approval as opposed to those who merely witness the signatures, notarizing signatures should be applied to the *encrypted* contents of the object, in such a way as to confirm the presence and authenticity of the previously applied digital signatures.

For the above reasons, the following additional globally-defined integrity categories or attributes will be required:

NON-REPUDIATION:

> This attribute implies (in the case of objects) or requires (in the case of subjects) a verifiable absence of modification of the object(s), plus non-repudiation (digital signature) protection of the contents of the object; the date, time, and place of the object's creation; and the identity of the signing user(s). NON-REPUDIATION requires that the originator of the document explicitly consent to sign the document to indicate his approval of its contents. Note that a given user may sign a document in different ways, depending on the user's *role* that is involved. Multiple users may sign or be required to sign the document in the case of a contract or approval list, as specified by the Discretionary Integrity Controls. The signature of the TCB, together with the trusted date/time mechanism, are sufficient to protect against a retroactive claim of compromised key(s) and a back-dated forgery by the signer(s).

NOTARIZED:

> This attribute requires or implies the presence of both the digital signature of the signing user(s), plus the digital signature of one or more independent witnesses. The NOTARIZED option differs from the NON-REPUDIATION option in that the witness is not approving the contents of document itself, but only witnessing the voluntary digital signatures of the other signers.

In general, before a user's digital signature will be accepted by another individual, the user will be required to sign a notarized Affidavit of Legal Mark wherein he or she agrees to be legally bound by his digital signature, perhaps subject to certain caveats. This affidavit announces the individual's public authentication key and is digitally signed, thereby proving that the

59. Without notarization, one party to a contract could attempt to renege on an agreement by convincing a judge that his digital signature was a back-dated forgery made possible by an alleged after-the-fact compromise of the signing party's digital signature key. The TCB's notarization attests to the date of the signature, and makes such a claim impossible in the absence of collusion with the DAA, who controls the date/time function in the TCB.

individual knows the associated secret signature key. The affidavit also contains the individual's public encryption key, so that anyone sending information to that user can be convinced that it is going to the intended receiver. Any other users should carefully examine the caveats, expiration date, etc., contained in the affidavit and should check any digital signature using the public key contained in the affidavit before accepting the validity of a particular document. Finally, the affidavit must be notarized, either by the electronic signature of a Notary Public (who must in turn be authenticated, and so on up to the head of a chain of authority deemed acceptable by the sponsor or owner of the TCB, e.g, the DAA) or with a traditional paper document, both to ascertain the integrity of the digital signature itself and to provide assurance that the individual whose digital signature is affixed to an object has been satisfactorily identified and duly authorized to sign a document.

4.10 Human-Readable Output

The above controls have dealt with the problem of ensuring the integrity of computer-readable information, but we are left with the problem of preserving and attesting to the integrity of the output on the video display and on the printed page.

The video display problem can be solved by treating the display hardware as part of the subject created on behalf of the user. If the user logs on with an integrity-read-limit which includes the attribute "TRUSTED-DISPLAY," any data which is to be displayed must be filtered through a program that has been certified as meeting the criteria established. That is, if any escape sequences are supported might alter the form of data presented on the screen, those escape sequences must involve only control characters which are outside of the normal alphabet. Furthermore, any TP which has the TRUSTED-DISPLAY attribute must scan all data to be presented and flag any such escape sequences. Finally, any TRUSTED-DISPLAY TPs must be certified to present all of the data, and only that data, which is intended to be presented.

The TCB's trusted path mechanism will be used to unambiguously inform the user of the integrity level of the display. **Note that a trusted path is a requirement for B2 systems and above — a B1 system will not satisfy this requirement unless this additional capability is included.**

The trusted printer output problem can be solved in the same manner, by supporting the "TRUSTED-OUTPUT" integrity attribute. If the user logs on requesting this integrity attribute, all programs which write output to the printer must be trusted to transform the computer-readable files into hardcopy output "properly." The trusted path will be used to inform the user of the integrity level of the output, and in addition, the banner page that is printed before and after any printed output will also inform the user of the integrity level of the output.

However, the trusted facility manual will have to address what types of printer can be connected to the system, and/or what precautions are to be taken to prevent a Trojan horse attack on the printer itself. The user will also be responsible for ensuring that the TRUSTED-OUTPUT attribute is invoked before printing any high-integrity output such as checks.

Because the TRUSTED-DISPLAY and TRUSTED-OUTPUT attributes may be application dependent and hardware specific, they are not defined as globally-defined integrity attributes.

5 CONCLUSIONS AND RECOMMENDATIONS

5.1 Reprise

The previous sections have dealt extensively with the problem of ensuring the integrity of information as it is processed, transmitted, and stored within a potentially unsecure network of interconnected TCBs. In concentrating on the problem of detecting unauthorized modification that may occur while the information is being transmitted or stored outside of a TCB and highlighting the shortcomings of the TCSEC and Clark and Wilson's paper in not dealing with these problems, we should not be unappreciative of the contributions made by those who have gone before. Indeed, if it were not for the contributions of people like Marshall Abrams,

Len Adleman, Jim Anderson, Al Arsenault, David Bell, Ken Biba, Earl Boebert, Dennis Branstad, Sheila Brand, David Clark, Don Coppersmith, Dorothy Denning, Whitfield Diffie, Virgil Gligor, Marty Hellman, Steve Kent, Len LaPadula, Steve Lipner, Mike Matyas, Carl Meyer, Ron Rivest, Marvin Schaeffer, Roger Schell, Adi Shamir, Bill Shockley, Gus Simmons, Steve Walker, Willis Ware, David Wilson, John Woodward, and many others before them, we would not be where we are today.

In particular, the virtues of the TCSEC should not be ignored, despite some of its present faults. Security and integrity must go hand in hand, and in particular, the system architecture requirements of the TCSEC for Class B3 and A1 systems, namely the need for the rigorous decomposition of function and the requirement for significant system engineering to minimize the complexity of the TCB and exclude from the TCB modules that are not protection-critical, are perhaps even more important from the standpoint of integrity than they are for security. In addition, the use of a digital signature mechanism for integrity implies the need for a secrecy mechanism within the TCB, so that the user's secret signature key cannot be compromised.

Although the emphasis in this paper has been on the defensive detection of unauthorized modification of information, whether it occurs within the TCB (perhaps by a virus that has somehow slipped into the system) or outside the TCB, it is obviously better to prevent such damage if possible. For this reason, the Mandatory Integrity Controls were formulated in such a way as to *prevent* a low-integrity subject acting within the TCB from modifying, deleting, or renaming higher-integrity objects within the TCB, although the Mandatory Integrity Controls can only *detect* modifications that happen outside of the TCB. In addition, although conventional Discretionary Access Controls with read/write permission, whether of the simple self/group/public type or a more comprehensive Access Control List, were found to be inadequate as a basis for a strong integrity confinement policy, they are still useful in supplementing the Mandatory Integrity Controls to prevent the modification, destruction, or renaming of an object by a user who has the necessary integrity certificates but was not authorized by the owner of the object.

In the telecommunications environment, the usual node-to-node transmission checksums and Go-Back-N or Selective Reject protocols are still desirable to efficiently correct accidental errors, and host-to-host or end-to-end packet sequence numbers and other techniques will still be required in order to minimize the need to retransmit an entire file if an error is eventually discovered. Protection against the physical interruption of communications (cutting the line) or the flooding of the network with unauthorized data (jamming) can be addressed through mechanisms such as redundant communications, protected wireline distribution systems, and anti-jamming techniques such as spread-spectrum communications.

5.2 Conclusions

The integrity requirements for both military and commercial applications can be satisfied by the use of Mandatory and Discretionary Integrity Control mechanisms that are the duals of the Mandatory and Discretionary Access Controls presently defined in the TCSEC, and can be implemented through the use of integrity labels, cryptographic checksums, and digital signature mechanisms, using a Trusted Computing Base (TCB) that provides both security and integrity protection:

Mandatory Integrity Control: Before a data object can be accepted by a process, a convincing argument must be made that an object has not been modified since it was created. This argument requires the verification of a cryptographic checksum and the digital signature of the originating TCB each time the object is accessed. If the verification fails, the integrity of the object is set to null.

Certain processes, called Transformation Procedures (TPs), must be certified by an approval authority with respect to a specified integrity domain. TPs must either be required to accept or believe only those objects that have already been screened with respect to a specified integrity domain by some other program or process, or else they must be prepared to validate (accept or reject) such inputs according to

application-dependent syntactic and semantic rules; *i.e.*, with respect to some integrity domain. More generally, an object is to be trusted, accepted, believed, or obeyed if and only if it was produced by set of trusted processes, which in turn accepted only certain trusted data inputs.

The Biba integrity policy model is extended to include the concept of a multilevel subject which is constrained to operate within an integrity-read-limit and an integrity-write-limit that include a hierarchical integrity level and a set of integrity categories, where the integrity categories correspond to the integrity domains for which a subject was certified. Such a policy can be used to constrain subjects from accessing (reading or executing) objects of lower integrity than a specified minimum level of integrity and from writing (creating, modifying, or deleting) objects of greater than a specified maximum level of integrity; thereby implementing the necessary constraints upon TPs. However, within those limits a TP may be accredited by some authority as carrying out whatever trusted process is required by the application.

Discretionary Integrity Control: All users of the system must be enrolled into the system and uniquely identified before accessing any resources.

All users must have their privileges (granted to them by someone of higher authority) authenticated before they are allowed to perform any action or any data produced by their actions may be accepted or believed.

Data introduced into the TCB or created or modified within the TCB by a subject (TP) under the control of some authorized user may be required to be explicitly associated with that user, who may be held accountable for his actions. An optional digital signature mechanism is provided to ensure that the user's signature was of his own volition, in order to establish intent.

The system must be capable of ensuring that the input to certain processes (TPs) may only be accepted (believed) if one or more trusted individuals approved that input. In some cases this may require excluding certain individuals from initiating a process, and/or requiring two or more trusted individuals to initiate a process, or only accepting or believing the input data subject to those restrictions, in order to enforce the principle of separation of duty.

Additional Controls: Additional controls are required to ensure proper sequencing and program confinement, as well as auditing and journalling capabilities that are suitable for the networking environment. A trusted date/time facility is required, and the standard audit trail is enhanced to more specifically address integrity concerns. Because it may not be possible for the recipient of some object to determine how that object was created by accessing the audit data base, a portion of that information, called the *pedigree*, can optionally be required to be present in the integrity label of an object. Statements *about* the object can be added to the *provenance* of the object and digitally signed by an authorized user.

Consideration was given to the possibility of integrity covert channels, and it was concluded that the primary risk is in the area of denial of service. In this connection, Yu and Gligor's[60] approach to a formal specification of service and user agreements using temporal logic seems quite promising, and may provide a means of tying the denial-of-service problem to integrity.

The above policies and mechanisms implement an integrity policy that is compatible with the concepts and rules originally proposed by Clark and Wilson, but is more flexible and more complete.[61] Such a policy appears to be easily implementable in an end-to-end, user-to-user networking environment of personal computers

---

60. Yu and Gligor, *ibid.*.

61. Clark and Wilson's most recent paper on the subject, "Evolution of a Model for Computer Integrity," was presented and informally distributed at the 11th National Computer Security Conference and at the NIST Invitational Workshop on Data Integrity, January 25-27, 1989, in Gaithersburg, MD. In it, they broadened some of the concepts of internal consistency of data that were introduced in their earlier paper. They now talk about Prevention of Change, Attribution of Change, Constraint of Change, and Partition of Change. The concept of change logs and integrity labels on data is discussed, along with enhanced user authentication and dynamic separation of duty. They note that "The most sophisticated form of logging, which remains rather speculative, is the labelling of data to indicate comparison with integrity domains." Their current position seems to be gratifyingly close to the policies and mechanisms proposed herein.

and workstations, and it should be highly useful for both commercial and military applications. It should be particularly useful for applications such as Electronic Data Interchange (EDI), where a large number of disparate organizations wish to exchange purchase orders and similar information across an untrusted network, and there is no central authority to coordinate all of the security policies and implementations of all the different organizations.

5.3 Recommendations

I would now like to offer some personal observations and suggestions regarding national policy, and what we in the computer security community should do about the integrity issue.

The publication of the original paper by Clark and Wilson has served to crystalize the thinking of the computer security community about integrity. Since then, the papers presented in draft form at WIPCIS in 1987 and published at the Oakland, Baltimore, and Orlando conferences during 1988 by Lee, Karger, Shockley, Clark and Wilson, Johnson, and myself, plus the discussion at the NIST Invitational Workshop on Data Integrity in January of 1989 have served to confirm the general validity of an approach to integrity that makes use of the Biba model extended to include multilevel integrity-trusted subjects to address the integrity confinement problem within the TCB, together with the use of cryptographic checksums and digital signatures to detect modification of information while it is transmitted or stored outside of the TCB.

While there will no doubt be additional comment regarding the specific details of this paper, I believe that it represents a reasonable consensus as to an approach that is "do-able" using today's technology for a Trusted Computing Base, and that in fact the amount of effort required to implement these concepts on Gemini Computer's GEMSOS, the Honeywell SCTC LOCK effort, Key Logic's KeyKOS, and several other high-assurance kernel developments in progress should not be all that great.

From the standpoint of the vendors and system integrators who will be asked to produce, install, and support these systems, I believe that the first priority should be for the Government (specifically the NCSC, with the cooperation of NIST) to produce a revised TCSEC that explicitly addresses the integrity issues in a coherent fashion. I believe that a concerted effort in this direction is required because of the increasing problem of computer viruses in our society. For that reason I would like to see a target date for a working draft of the revised TCSEC that is no later than January 1990, with final publication scheduled no later than June 1990. If necessary, additional funding should be provided to NIST and/or NCSC to address this problem. **A business-as-usual approach will not be adequate.**

In this regard, I strongly suggest that an TNI-like "interpretation" of the TCSEC for integrity is *not* the best way to proceed. The basic problem is that the TCSEC does not address the fundamental issues of integrity, and trying to "interpret" them into existence risks basing a complex structure on a shaky foundation. In addition, I feel that dual ratings or a "separate but equal" status for integrity as opposed to security would result in fragmenting the marketplace, and would be highly undesirable.

Instead, I would suggest that integrity be explicitly integrated into the TCSEC as a new requirement for TCBs of Class B3 and above, and that developers of systems targeted for lower ratings be encouraged to support the functionality of the additional integrity features, if not necessarily the assurance of those higher classes.

Such an approach would provide a clear and unambiguous target for vendors to address, whereas alternative systems, separate ratings, etc., would be likely to fragment the marketplace and end up producing few if any useful systems. By combining a high degree of assurance for both secrecy and integrity the result should be an increased demand for high-assurance TCBs, which would benefit both the vendors and the user community. Because there are presently no systems that have been evaluated at the B3 level, and those that are in the process could easily be adapted to conform, this approach would produce the greatest overall improvement with the least amount of incompatibility with previously published ratings. Only the Honeywell SCOMP would have its A1 rating potentially invalidated by this approach, and because relatively few SCOMPs have been sold that

problem could be handled by a footnote in the published ratings.

In addition to publishing a revised TCSEC which addresses integrity, it would also be useful if the NCSC would sponsor an effort to integrate the Bell-LaPadula and Biba models into a combined security/integrity formal policy model along the lines presented in this paper, so that the TCB vendor community (which is not necessarily "into" the business of formal policy model development and proof) would have the requisite technical underpinnings for a B3 system in place.

Another issue that must be addressed in this context is the question of exportability. At the present time, it is understood that systems of B3 and higher assurance cannot be exported, or that special licenses are likely to be required. It would be one thing if only specialized systems intended for the military were to fall into this category, but if the introduction and formal evaluation of high-assurance integrity features into systems that are intended to be commercially useful would mean that those systems cannot be exported, then it can be predicted that virtually every major computer manufacturer and operating system vendor in the United States would say "no thanks" to such a proposition because of the volume of their international business. It is simply not possible to develop and maintain two different systems which support essentially the same user community.

The message to the Department of Defense, the Department of State, and the Department of Commerce must therefore be of "Read My Lips" clarity — if we are to come up with an effective solution to the problem of computer viruses, we *must* overcome the export problem. Otherwise, the Japanese and/or the Europeans are likely to solve the problem for us, and then we will be forced to *import* the computer software (and perhaps hardware) that we need. Such a dependency would clearly not be in the best interest of the U.S. from an economic, diplomatic, or military point of view.

Finally, while I would not like to see the publication of a revised TCSEC held up while the computer security community debates what a Class A2 TCB should contain, it is increasingly obvious that trusted compilers and other program development tools are badly needed, and that such systems must be supported by a TCB of the type described herein at the B3 level or above.

I would therefore recommend that the Department of Defense Ada Program Office initiate R&D programs as necessary to develop a certified Ada compiler which is written in Ada and can compile itself, one that will run on a B3 or higher TCB and produce code that is compatible with such systems. A trusted Ada run-time environment that is compatible with a B3 or higher TCB is also required. In this connection, it is recommended that steps be taken to eliminate the possibility of a compiler virus of the type described by Ken Thompson and discussed previously, including the use of cleared and trusted programmers to conduct a rigorous Independent Verification and Validation (IV&V) of the compiler and run-time routines, including compiler source code review, independent compiler generation and object module verification, an informal reasonableness review of the compiler-generated assembly language listing vs. the source code, and a comparison of the generated object module with the assembly language listing. Finally, it would be desirable if a suitable subset of secrecy-trusted and integrity-trusted Ada run-time routines could be developed and integrated into an A1 TCB (including the formal proof process), so that multilevel-secure Ada routines can be written for applications like Automated Message Handling Systems, trusted downgraders, etc. Eventually, the use of such tools, running on at least a B3 system that includes the integrity provisions discussed in this paper, should be mandated for the development process for a Class A2 TCB.

If the steps I have recommended are taken, it is reasonable to expect that solutions to the current virus problems and other integrity issues will begin to come under control by the mid-to-late 1990's. The question is therefore not whether we can afford to develop these kinds of systems, but rather whether we can afford to wait that long.

SECURITY CLASSES AND ACCESS RIGHTS IN A DISTRIBUTED SYSTEM

R W Jones

ICL Defence Systems, International Computers Ltd., Eskdale Road, Wokingham, Reading, RG11 5TT, U.K.

SUMMARY

A model is described for the management and use of a network of computing resources in which the control of security is to be explicit. It is intended to be useful in military and civil applications. A generic form for a **security class** is defined. Using this form a total security class for a particular system may be defined, in terms of **attributes** with agreed meanings in the environment of the use of the system. The total class contains **subclasses** which are used to **classify** all the **components** of the system. A subclass is a class which is more specific than its containing class. For each class a set of access rights is defined. An access right is a set of operations which may be performed upon components classified by the class of the access right. Components include data items, computing resources and end users. An **active component** (e.g. a computing resource) is one which may operate upon other components. Each active component has a **clearance** which is a class with associated **access rights**. A **security policy** is defined as a set of rules which associates classification classes with maximum permitted clearances.

Communication between **security domains** is discussed in terms of the model, as is secure system construction. The model is related to other published models and standards.

Notes on the Paper 'Security Classes and Access Rights in a Distributed System'

As the result of the presentation of this paper at the Data Integrity Workshop and from subsequent conversation with a number of people, I have become aware of some mistakes which impeded understanding of the ideas in the paper. I have therefore changed it in some places and describe the changes in more detail in the following notes. There are some cases also where comments might have caused me to describe things differently (in particular to use different terminology) but where I have left the text as it was, thinking that a change might be confusing. I have written notes also on such cases. There is also some text which I omitted from the version submitted to the Gaithersburg workshop to shorten the paper.

1. (Section 2). The terms 'security class', 'classification' and 'clearance' have been criticised as having military connotations. I have not changed them. The term 'security category' has been suggested instead of 'security class' and 'remit' instead of 'clearance'. In retrospect I believe I should have avoided the term 'classification' or any substitute and described a component as <u>being</u> of a security category (which determines the operations which may be performed upon it) and as <u>having</u> a clearance (or remit) which determines what it may do).

2. (Section 2). In the original paper an access right was described as a component. It no longer is, as this seems clearer.

3. (Section 3). Some people have commented that the model I describe calls for capabilities to describe access rights, but not access lists. There is no such intention. The model is intended to be understood at a more abstract level. Both capabilities and access rights are possible mechanisms for representing rights in an implementation.

4. (Section 3). In the original paper I tried to use Extended BNF to describe both the generic form of a security class and for specific security classes and their relationship to each other. Some mistakes crept in and, therefore, some misunderstanding. In the version of the paper printed here I have tried to prevent the misunderstanding by using BNF only for the first purpose and correcting the mistakes.

5. (Section 3). Some of the points made later in the paper could have been put more simply if a the term 'subclass' had been defined to include the containing class.

6. (Section 4). It is not intended that the mechanisms described here be the only means of distinguishing individual components, rather that they be used to distinguish sets of components which need to be distinguished for reasons of security.

7. (Section 4). If access is to be allowed the following must be true:

i) there must be a class which is contained in both the accessor's clearance and the target component's classification,

ii) that identified class must have rights which are contained in the accessor's clearance and they must have been supplied to the accessor and not subsequently withdrawn (see section 5.2).

8. (Section 4) It is not clear in the paper when rights defined for a class should apply to its subclasses and vice versa. I believe it is necessary to be able to define rights which apply strictly to a class or to defined subclasses or to a class and its subclasses. More consideration is needed of this.

9. (Section 4) One criticism made of the paper is that it is not obvious that a right which appears in a clearance should also need to be 'supplied'. The motive for the extra flexibility is to enable components other than the original creator of a resource to provide and withdraw rights, subject to predefined constraints. There is an analogy with mandatory and discretionary access control in military systems.

10. (Section 5.2). It may be useful to be able to authorise a class of components.

11. (Section 6.3). This section did not appear in the paper as presented at Gaithersburg.

R W Jones

21.6.89

Security Classes and Access Rights in a Distributed System

1. Introduction

1.1 Aims and Summary of the Paper

The paper describes a general design for a secure system which uses a potentially distributed network of computing resources. The design is based on the idea of a security class, a generic form for which is defined in the paper. The generic form provides a notation which may be used to describe specific security classes for an individual system. The paper describes a model which applies both to the secure creation and management of a system and to its secure use. It uses and builds on ideas described in ref. 1 and ref. 2.

The aims of the paper are to:

i) describe a general design for a secure system (covering civil and military needs) and the means of tailoring it to a particular system,

ii) provide a notation which enables the requirements of a secure system to be related to its design,

ii) clarify terminology and ideas in secure system design.

The paper is organised as follows.

Section 1.2 describes the context of the paper.

Section 2 defines some terms.

Section 3 defines a security class and explains its use.

Section 4 uses the idea of a security class to introduce access rights, classifications and clearances, these being used to describe how security relates to the components of a system.

Section 5 describes specific access rights which are useful in most systems. They include those needed for:

change of security class (applicable to identification and authentication),

creating and controlling the components of a system,

delegation of authority.

Section 6 compares the model of this paper with other published models and standards.

Section 7 provides examples in terms of the paper.

Section 8 defines security policies in terms of this paper.

Section 9 defines security domains in terms of this paper.

Section 10 describes secure system construction.

Section 11 draws conclusions and considers future work needed.

1.2 Background

In non-automated systems where people have to handle confidential information it is customary to give the information a classification, for example, 'company confidential'. The people are then also classified according to the amount of trust that should be placed in them. This takes into account both the likelihood that they will abide by the rules which safeguard the information and their need to know it in order to perform their function properly. The people in such an environment, therefore, have both a classification, based on their own trustworthiness, and a clearance which describes the kinds of information they may handle.

There is a management function for the system which decides how information and people are to be classified, what these classifications mean, in terms of the characteristics of the people and the information and objects they manipulate, and which classes of people may have which clearances. The management also provides the classifications and clearances based on these rules and changes and deletes them as necessary. The amount of formality attached to this management function depends upon the environment. It may be very informal in a small commercial firm and is very formal in the armed forces (formal in the sense that the processes involved are performed according to explicit rules).

If we now consider that a network of computer resources is to be used as part of the system which handles information securely we see that we need mechanisms which perform the equivalent of both the management and the operational functions described above. They need not mimic them exactly but it must be clear that they allow the system to be created, managed and operated securely. If we provide the mechanisms which cause the system to operate securely but provide them without using centrally provided mechanisms, for example by building access control checks into application code rather than by installing them into supporting control software, we are in the position of the small firm mentioned above. Since the computer system itself allows us to describe rules precisely and ensure that they are obeyed we have the opportunity to make explicit both the provision and the operation of the security mechanisms. This paper attempts to contribute to this by describing a model on which mechanisms can be based which perform both the management and operational security functions.

The context, described more precisely, is a community of computer installations which communicate using telecommunications and agreed standard protocols (for example those of ref.3). Those standards formalise a computer installation which communicates according to its standards as an 'open system'. They allow a single open system to support a number of software entities and allow any entity in an open system to exchange messages with an entity in another open system. This is illustrated in figure 1.

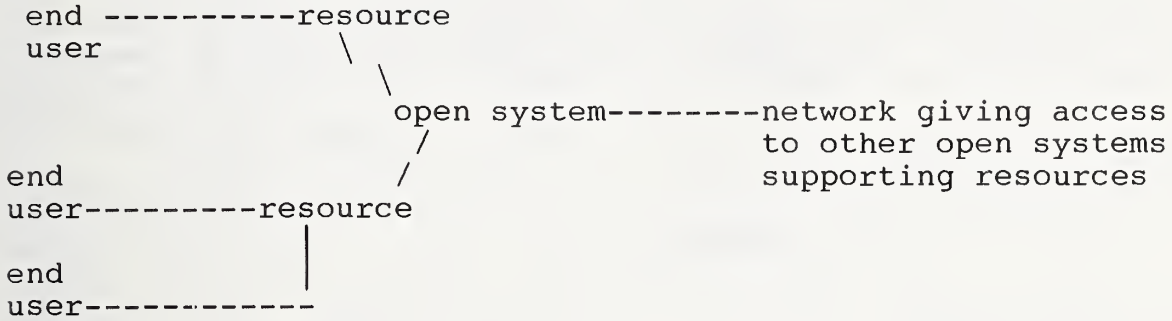


figure 1

2. Terminology [see note 1]

In order to discuss the model a set of terms is now defined. The aim is to depart from accepted usage only to avoid misunderstanding.

An **end user** is a person or entity which is not controlled by a resource of the network, but which can communicate with a resource. An end user may be, for example, a person sitting at a terminal.

A **resource** is a logical part of a computer system which can store data items, which can communicate with other such resources or with end users and which has a current state which determines its actions.

A **data item** is an item of data which is created with a defined security class and the parts of which necessarily have the same class. The term **message** is used to mean a data item when it is sent from one resource to another.

A **channel** is the route by means of which a resource communicates with another resource or with an end user.

An **access right** is a collection of operations which is defined for a particular security class. The holder of an access right has permission to perform those operations upon components which are described by the class in question. Where there is no chance of confusion the term 'right' is used instead of 'access right'.

There may be more than one access right for any given class in order to divide operations into groups and control access to groups individually.

A **component** is any entity which has a security class. End users, resources, data items and channels all have security classes and are therefore components.

Components may be **active** or **passive** depending on whether they can hold access rights. End users and resources and channels are active components. Data items are passive components. A channel may only hold access rights which enable it to transmit messages.

The components named above may be grouped diagrammatically, therefore, as follows:

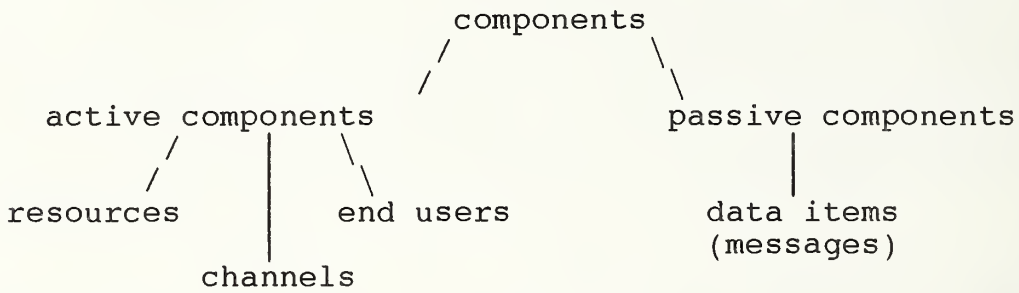


figure 2 [see note 2]

Refs 1 and 2 described a model in terms of rights which enabled access control rules to be enforced. The present paper builds on these concepts and adds to them the idea of security classes. These are compatible with and include the concepts used in data classifications and clearances in military systems and, with some modifications, in civil systems. Where there is no chance of confusion the term 'class' is used rather than 'security class'. An exact meaning of the term is given in section 3, which also introduces other terms for the purpose.

A **classification** is the security class of a component which determines whether or not it may be operated on by another component. Every component has a classification.

A **clearance** is the security class and associated access rights associated with an active component which is used in determining whether or not the component may operate on another component. An active component has a classification and a clearance. A passive component has a classification but no clearance. The relationships among a clearance, a classification and an access right are described in section 3.

The term 'system' is used to mean a communicating set of components whose access to each other is controlled by mechanisms which take account of their security classes. There is no implication that the components are dedicated to work which is all interrelated, although that is likely to be true in practice.

The term 'subject' is used to refer to an active component which performs an operation upon another component.

The term 'object' is used to refer to a component (active or passive) upon which an operation is performed.

3. Security Classes [see note3]

If we consider once more the non automated secure environment we can see that one way of stating rules of access is to have a unique identity for each object to which access is to be controlled and to provide each person with a list of the objects he may access, together with, in each case, a list of the operations he may perform. In order to provide clearances to the people, they themselves are treated as objects and some manager has the list of people to whom he may give a clearance or whose clearance he may amend. The procedures needed, therefore, both to operate and to manage the secure system in this simple way are similar. For each person who may perform operations, a list of the permitted objects is needed and, for each of those, a list of the operations which may be performed. This applies both to operation and management.

In a more complicated system clearances are expressed in terms of attributes of the objects accessed for two reasons:

i) it enables objects to be grouped so that clearances are more concise,

ii) the attributes of the object's classification have meanings which have a relationship to the attributes of the accessor's classification (e.g. an object with the attribute 'payroll information' is to be available only to someone with the attribute 'member of the payroll department').

The definition of a security class which now follows is based on attributes. The meanings of individual attributes depend upon and are agreed for a particular system. An attribute may be sufficient to define the security class of an object uniquely or may describe one of its characteristics.

Extended Backus-Naur notation is used in the following definition, the meta-symbols having the following meanings: [see note 4]

'::=' means 'is defined as',

'|' means 'or',

';' terminates a syntactic rule,

single quote symbols delimit items which appear as written in the rules defined,

'*' indicates one or more occurences of the item to its left.

a string of (possibly hyphenated) letters is an identifier which is either defined by one of the rules or is described informally.

The generic form of a security class is then as follows:

class-definition ::= class-name ':' lower-limit ',' qualifier*;

class-name ::= name;

lower-limit ::= integer;

qualifier ::= class-name | attribute;

attribute ::= name;

where $1 \leq$ lower-limit $\leq$ number of qualifiers;

An attribute is a name declared for the definition and control of the security of the system, as described above.

A name is a sequence of characters generated according to some rule that ensures that each name generated to describe the system's security is distinct from all those generated previously for the same purpose.

A lower limit defines that at least that number of the defined qualifiers is present in the class being defined or in one of its subclasses.

Two particular values of 'lower-limit' are worth distinguishing:

i) the value 1, where the class describes a collection of qualifiers, any one or more of which may be present.

ii) where the value of 'lower limit' is equal to the number of qualifiers; this describes a class in which all the qualifiers are present; a simple case of its use is where the qualifiers are attributes and describe all those which must always be present to define the class.

A subclass is derived from the definition of a class

i) by selecting n of its qualifiers such that $b \leq n \leq q$, where

b is the lower-limit of the class from which the selection is made and

q is the number of qualifiers in the class from which the selection is made

and

ii) by selecting a lower-limit for the subclass such that b $\leq$ lower-limit $\leq$ n,

with the restriction that the subclass cannot be identical with the class from which it is derived. [see note 5]

A subclass is also a class. If a class S may be derived from a class C then C will be called the containing class of S.

A subclass may also be derived by replacing one or more of the classes used in the definition of a class by a subclass. See figure 3 and the subsequent explanation.

The intention behind the definition of a class may be illustrated by a small example. Let us suppose that a commercial firm creates documents, access to which is to be controlled, depending on the trustworthiness and duties of the accessors. The attributes 'confidential', 'pay' and 'plans' are used to classify the documents so that security controls may be applied. Some documents about pay are confidential; some are not. Some documents about plans are confidential; some are not. Some documents are simply classified as confidential and these are not concerned with either pay or plans. The people who may use the documents are cleared in terms of the documents they may access as follows:

all documents,
all confidential documents,
confidential documents not concerned with pay or plans,
confidential documents about pay or plans or both,
only non-confidential pay documents,
only non-confidential plans.

The classes needed to express these needs is are as follows:

```
all : 1, all-conf, subjects;
all-conf : 1, conf, conf-subjects;
conf-subjects : 2, conf, subjects;
subjects : 1, pay, plans;
```

It is illustrated by the following graph:

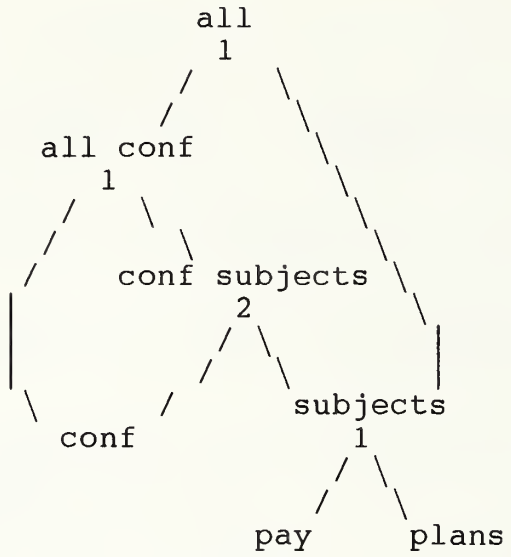


figure 3

[The graph makes it clear that the figure 1 could be replaced in each case by the 'inclusive or' operation, meaning that one or more of the qualifiers must be present, and the figure 2 by the 'and' operation, meaning that all the qualifiers must be present. The syntax could be rewritten to match this and would be trivially different. The forms where the lower limit lies between 1 and the number of qualifiers would be more cumbersome to express.]

It will be seen from the above example that the derivation of a subclass produces a more specific description in terms of security. The class 'conf-subjects' has three subclasses illustrated graphically below.

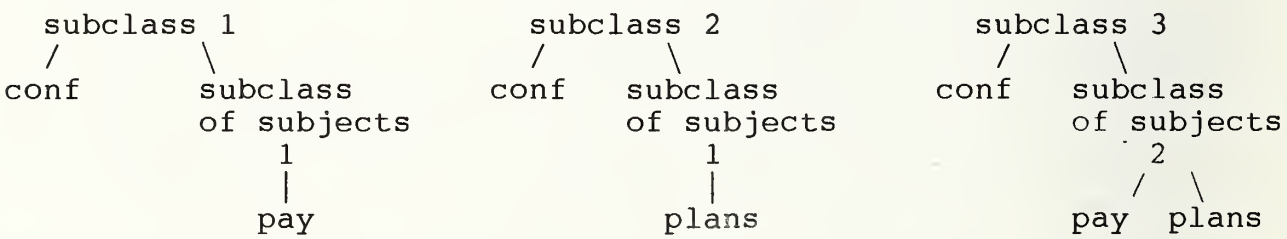


figure 4

If a class from which subclasses may be derived is used to classify a component it means that the component concerns any one or more of those subclasses. It may be used also to describe the fact that an active component has a multi-class clearance. This is described further in section 4.

A selected subclass may be equivalent to one defined explicitly and used as a qualifier in the definition of the class from which the subclass derives, for example in selecting the class 'authorised' from the class 'accessors', where 'accessors' is defined as

accessors : 1, authorised, unauthorised;

where 'authorised' is defined as;

authorised : 1, Alice, Bob, Charles;

and 'unauthorised' is an attribute defined for the system.

It may be implicitly define, for example the class obtained by deriving the subclass '1, Alice, Bob' from 'authorised'.

A class is defined explicitly so that it may be used as a classification or clearance of a component and in order to be able to define access rights and operations for the class.

Two classes may contain the same subclass without themselves being identical or one of them containing the other, for example the classes

'1, Xusers, unauthorised' and '1, Yusers, unauthorised'.

4 Classifications, Clearances and Access Rights

A class is defined for the total system. Each class used in the system is a subset of this total class.

For each class of the system a set of access rights is declared. Each of these access rights defines one or more operations for that class. (In refs 1 and 2 a particular right applies to a single defined target resource. It is generalised here to apply to a class, which may, in a particular case, be a class which is used to classify only one component.)

Each component of the system has a classification which is a class. The classification defines the operations which may be performed upon the component (those of its class) and the kind of component it is in terms of security (because the attributes of its class have an agreed meaning). **[see note 6]**

Each active component also has a clearance which is a class. The clearance defines the operations which the component may perform

upon other components (those whose classifications or subclasses of whose classifications appear in its clearance). The clearance therefore defines the trust which is placed in the component which possesses it. This is illustrated in figure 4. **[see note 7]**

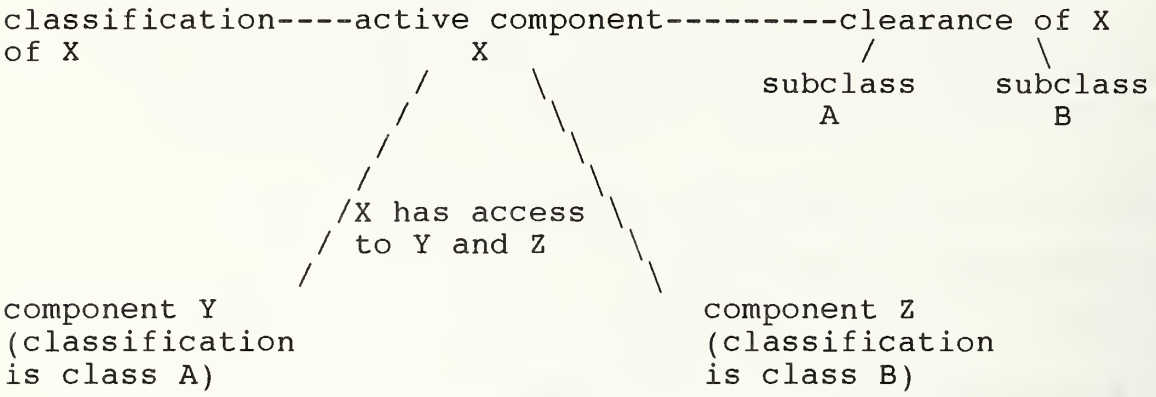


figure 4

It is necessary to provide different accessors with different rights in respect of the same component accessed. Two methods of doing this are described here.

i) The class used for the classification of the component to be accessed may be a subclass of more than one other class. Each of the containing classes has a set of rights which applies to its subclass. The union of these sets of rights form the total set which applies to a component classified by the subclass. This is illustrated in figure 5. **[see note 8]**

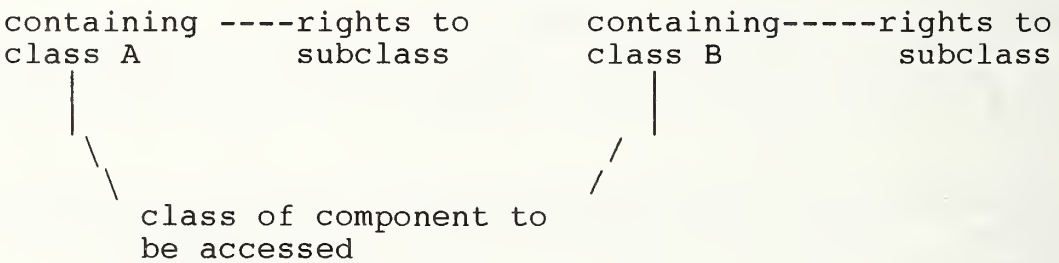


figure 5

(In an implementation of the design operations other than the union of the sets of rights may be used for efficiency; for example, the subclass may define all the applicable rights and the containing classes those which are not available via that containing class).

ii) The clearance of a resource defines for its most general class and for its subclasses individually the access rights which it is allowed to possess. These are the total set or a

subset of those defined for the class. For some of its subclasses none may be allowed.

Notionally the second method is unnecessary since one may define a containing class with the selection of rights needed. It implies that a class may be created as needed and associated with the clearance. For the purposes of this paper it is assumed that both methods are available.

By analogy with the idea of providing a clearance with a subset of the rights defined for its class one might allow the classification of a component to have a subset of the set of rights defined for its class, thus restricting the operations available to all accessors. Again this is notionally unnecessary since a class may be defined with the required rights. It may be useful in practice to avoid a large number of classes.

Consider a class defined as follows:

accessors : 1, Alice, Bob, Charles, unauthorised;

There are four subclasses of interest, namely those defined by the single attributes Alice, Bob, Charles; and 'unauthorised'. Let us assume that the first three of these more elementary classes are used to classify data from three authorised users of the system and that 'unauthorised' is a class which describes data from any unauthorised person who tries to access the system. The resource which may be accessed by anyone who approaches the system has 'accessors' within its clearance. It receives data from a user who wishes to use the system via a channel whose clearance is 'accessors'. The first message it receives therefore has the class 'accessors', representing the fact that it may come from any of those sources. The resource which is authorised to receive such a message then (in our example) engages in some procedure (authorised by one of its access rights) which allocates to the data received a subclass of 'accessors' (i.e. Alice, Bob, Charles or 'unauthorised'), representing the fact that it has authenticated one of the authorised users or recognised an attempt at a security breach.

There may be a resource between the accessors and the authorisation resource which is used to relay the accessors' messages but which is not authorised to classify the messages as coming from a particular user or as being unauthorised. This relay resource, like the channel which carries the messages, has the class 'accessors' within its clearance but has no rights which

apply only to the subclasses (see figure 6).

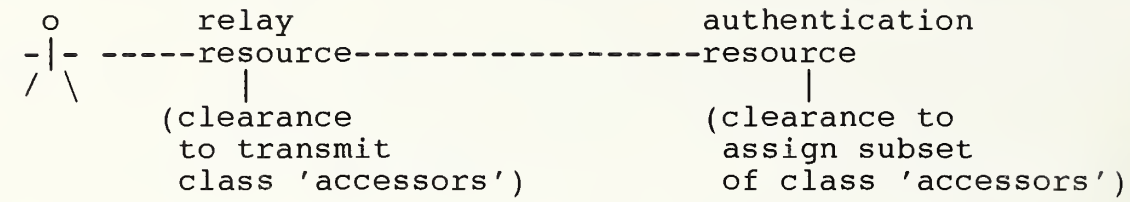


figure 6

Thus, by defining the most general class for which access rights may be possessed and by defining rights for that class and its subclasses a clearance may restrict its owner in both the generality and the particularity of the classes which it may handle.

When a resource is created it is given a classification and a clearance as part of the creation operation. The clearance must have the following properties.

i) It must conform to the security policy (i.e. it must be allowed for a resource of that classification, see section 8). It may be less than the permitted clearance in terms of classes and/or access rights.

ii) It must not contain class/access right combinations not possessed by its creator.

A clearance shows the access rights which the holder of the clearance is allowed to possess for each of the classes in the clearance. In order to use an access right it must actually possess it. The right must be provided, either by the resource's creator or by some other resource with the right to do so (see section 5.2). **[see note 9]**

These two features: allowing a clearance to be less than that imposed by the security policy and withholding the use of rights until they are supplied, allow local control of security subject to an overriding central policy. Since the ability to create a resource and provide it with a clearance may be inherited a hierarchy of control over the assignment of clearances is possible. At any level a creator may pass on a clearance which is less permissive than the most permissive which it is allowed to pass on.

If a clearance contains conditions which depend upon the state of the environment (for example that an access right is available only at certain times of the day) this is not catered for by the model as described so far. This kind of condition may be fitted into the general design by associating an attribute of a class in

a clearance with a procedure, whose result must be compatible with the corresponding attribute in the classification of the component accessed.

5 Specific Access Rights

In general the access rights and the operations which they provide are system specific and not defined by the model. Some, however, are generally useful and are described here.

5.1 Rights which apply to data items

right to change classification

This right provides an operation to change the classification of a component to a subclass or containing class of its current class. An example of change to become a subclass has been given in section 4. An example of change to become a containing class occurs when a resource sends a message. The classification of the message must be the class of the clearance of the channel by which it is sent. This may, for example, be a very general class in which sensitive data is indistinguishable from any other (in practice because the procedure which generalises the data involves encipherment).

5.2 Rights which apply to resources and end users

These are rights which enable resources to be created and controlled and which enable resources and end users to be provided with rights and to be sent data. They are very similar to the rights described in refs 1 and 2, (q.v. for more details). They are modified to allow for the introduction of security classes. They are as follows.

right to control resources

The right provides the following operation:

create resource This creates a resource of the designated class and with the designated clearance.

activate This makes the resource available to holders of rights to that class other than 'control';

suspend This makes the resource unavailable except to the holder of its control right;

change This changes the resource's code;

delete This withdraws the resource from service.

right to authorise resources and end users [see note 10]

the right provides the following operations.

supply This supplies to the target component a right held by the supplier; when the operation has been successfully performed both the supplier and the recipient possess the right; the right supplied must be for a class within the clearance of the target component. The supplier must be a resource.

withdraw This withdraws from the target component a right previously supplied by the same resource.

right to send messages to active components

This provides the following operation.

send message This sends a message to a resource. The message may instruct the resource to perform an operation upon data, in which case the sender must have clearance for the security classification of the data.

5.3 Rights which apply to classes

right to define classes

This provides the following operations:

define class This defines the syntax of the class for which the right is held. For a particular class only one active component may hold the right.

create attribute This creates a new attribute by generating a name. The attribute may be used when defining a class.

This right enables local, dynamically defined classes to be created. An example is where there is a need to create a file and exercise local control over access to it by other resources. A class may be defined for this purpose, where the class's only member is the file. In a more complicated case a resource may need to create several resources and enable each of them to create files and control access to them. This is achieved by passing on the right to define and create components classified by subclasses defined for the purpose.

6. Comparison with other Models and Standards for Secure Systems

6.1 Comparison with the Lattice Model of Information Flow

There are a number of significant differences between the model described here and the lattice model described in ref.4. They are

as follows.

i) A lower limit to the number of attributes of a class is stated in the model described here. This enables a clearance to be given for a class which is defined by a particular set of attributes without providing clearance for the classes which are defined by fewer of those same attributes. Thus clearance for the class '2, secret, pay' does not provide clearance for things described as just 'secret'; only for those which also relate to pay.

ii) Particular rights are defined for individual subclasses in the present model and may be null or restricted in some clearances to restrict the power of the component.

iii) In the present model a classification is distinguished from a clearance.

6.2 Comparison with the Clark-Wilson Model

Ref.5 describes a model for integrity. It has Constrained Data Items (CDIs) which are operated on by Integrity Verification Procedures (IVPs) and Transformation Procedures (TPs). The central ideas are:

i) that data items whose integrity is to be protected are labelled as Constrained Data Items (CDIs),

ii) that all CDIs are confirmed as conforming to a defined integrity specification by running Integrity Verification Procedures (IVPs),

iii) that any CDI can be operated on to change its state only by a Transformation Procedure (TP) which is certified to be valid for that CDI,

iv) that users, who must be authorised, are constrained to use only specified TPs on specified CDIs.

Rules are defined to ensure that the mechanisms and programs needed are certified as valid, that users are authenticated and that records are kept of the operation of TPs.

The model described in this paper provides a basic framework, using which a system which follows the rules of ref.5 can be constructed. This is explained as follows.

i) A CDI is a classification of a data item in order to specify and constrain the operations which may be performed upon it and corresponds to the classification of a data item using a security class.

ii) An IVP is a particular kind of operation in terms of the model described here which should be controlled by an access right which is available only to components with the correct clearance. It is

a particular feature of commercial systems that data is validated as a separate operation before it is operated upon. This is not a fundamental requirement of the model described here but the framework within which it can be done is provided.

iii) A TP is an operation, in terms of the model described here, which, like an IVP, is controlled by an access right. A system which is to preserve data integrity must insist that operations are performed, not only on data of the correct kind but in the right order. This may be done, using the model described here, by using a qualifier of the class which classifies the data to prescribe the sequence in the following way.

Let us define a class as follows:

CDIX : 2, X, sequence;
sequence : 1, unchecked, IVP, TPX1, TPX2, TPX3;

where X, let us say, means that the data belongs to project X and 'sequence' is used to ensure that operations are performed in the right order. 'X', 'IVP', 'TPX1', 'TPX2' and 'TPX3' are attributes of the class which describes the total system security.

An authorised user has a right to operate upon the subclass '2, X, unchecked'. The only operation the right allows him is one which performs the IVP operation. Part of the effect of this operation is to change the class of the data to a new subclass of 'CDIX', i.e. '2, X, IVP'. The user has a right to this class which enables the first appropriate TP to be performed. This similarly changes the subclass to '2, X, TPX1'. In this manner the operations are performed in the correct sequence. It may be noted that the correct order depends upon the individual TPs behaving correctly. They operate in an environment in which the whole class 'CDIX' is valid. If there is a risk that the user of the managerial function which has the right to provide the TPs will collude with the user the operations may be located in more than one resource and the resources managed independently.

iv) Users are constrained to use only specified TPs by providing a user with a classification and a clearance. The compatibility of the classification and the clearance are checked using the security policy (see section 8).

6.3 Relationship to the ECMA TC32/TG9 Security Model [see note 11]

The ECMA TC32/TG9 model is described in ref.6. It is a long document and it is not proposed here to explore in detail how it relates to the model of this paper. However, a set of security facilities are central to the ECMA model. This section therefore discusses them and relates them to the concepts described here. The facilities, with comments on them are as follows.

i) Subject sponsor facility

This sponsors the human user to the secure system during authentication and monitors his subsequent activities. There is no distinguished separate entity in the model of this paper which corresponds to this. A resource may be created by any resource which has that right. The subject sponsor is a resource fundamental to the type of system described by ref.6, which is, in this respect, a particular type of system which may be built using the model described here.

ii) Authentication facility

This authenticates human users and applications of the system. It is represented in this model by resources which have a clearance which enables them to change the classification of a message to the subclass which identifies an individual user. In this model there is no insistence either that a single resource should authenticate all users or that there be a class which defines an individual user. Such requirements are essential for many systems and are not precluded by the model.

iii) Association management facility

This sets up and maintains a secure association between entities of a secure distributed system which exchange information. In terms of this paper it is part of the functionality which is implied by the acknowledgement that the resources of the system may be distributed.

iv) Security state facility

This records the current state of the system which is relevant to its security. In terms of this paper it is part of the functionality which is implied to ensure that a system operates correctly according to the model.

v) Security attribute facility

This records the security attributes assigned to entities of the system. In terms of this paper its functionality is implied by the classifications and clearances of the components.

vi) Access Control facility

This associates attributes with entities and also uses them to check whether an access request is to be granted. In terms of this paper the first of these functions is achieved by creating a resource with a particular classification and clearance. The second function is implied when any access is attempted.

vii) Inter-domain facility

This controls access between entities in different security domains. Security domains are defined in terms of this paper in section 9 (q.v.).

viii) Security audit facility

This records information about the use of the security functions of the system. This function is not explicit in the model of this paper. The model makes explicit all operations relevant to security classes and access rights. In a practical system a security audit is provided by recording such operations.

ix) Security recovery facility

This facility is to enable a security administrator to take corrective action in case of a suspected breach of security. The model of this paper has nothing explicit to say about this. The assumption is that the required functionality is part of the definition of the individual system, which is built using the model described.

x) Cryptographic support facility

This provides the cryptographic functions needed for the operation of the system. In terms of this paper it is a possible mechanism for implementing the model.

In general the model of ref.6 is a prescription of facilities which are needed to provide a secure system of a particular kind which is likely to be frequently needed. The model of this paper is more general in that it includes systems which would not conform to ref.6. The model of this paper is, moreover, more abstract than ref.6 in that it does not prescribe how the functionality needed should be provided. It is intended that such a prescription should be given separately.

6.4 Relationship to other standards and guidelines for security

Ref.7, the US Department of Defense 'orange book', describes 'a uniform set of basic requirements and evaluation classes for assessing the effectiveness of security controls built into Automatic Data Processing (ADP) systems' (ref.7 foreword). The model described here aims to aid system design to provide some of its requirements in an explicit manner. It is concerned in particular with mandatory and discretionary access control in the terms of that document.

Ref.8 describes what is called a security architecture for the ISO reference model for open systems interconnection. It is therefore worth exploring how a model such as the one described here relates

to it. The concerns of ref.8 are:

i) to describe appropriate security services and mechanisms,

ii) to define where they may be provided in the reference model.

The security services described are grouped under the headings:

authentication,
access control,
data confidentiality,
data integrity,
non-repudiation.

All of these except the last relate directly to the model of this paper in that the open system interconnection standards may be used to provide communication services between remote resources. The security services may then be used to ensure that the communication is secure. The non-repudiation service has no direct relationship to the model. Its functions are directly relevant to system users in appropriate cases.

7 Examples of Use of Classifications, Clearances and Access Rights

Example 1 A class where all qualifiers are obligatory

Take the class defined as follows:

new-product : 2, confidential, cars;

We may imagine that information of this class concerns a new product which is confidential and relates to cars. Now a resource which had only the clearance 'conf', defined as:

conf : 1, confidential;

or 'vehicles', defined as:

vehicles : 1, cars, buses;

would be unable to hold an access right to the information.

A resource which had the clearance 'trusty', defined as:

trusty : 1, confidential, cars;

or

trusty : 1, confidential, vehicles;

with vehicles defined as above, would be able to handle information which was classified by either or both of the attributes, provided it held the appropriate access rights.

A resource which had the clearance 'new product' would be able, with appropriate access rights, to handle information about the new product but not other information concerned with vehicles or other confidential information.

Example 2 A class where all qualifiers are not obligatory

A data item is recorded in the system and, to be recognised as genuine, must bear the signature of at least two of five possible signatories. Assume that there is a resource to which such messages are sent which is able to decide if the message is properly signed. This resource has, as part of its clearance, the class 'signed or unsigned', defined as:

signed or unsigned : 1, signed, unsigned;
signed : 2, Alice, Bob, Charles, Don, Eliza;

where the peoples' names are the attributes used to classify the signatures and 'unsigned' is a class with a single attribute, used to record that a message is not properly signed (including the case where it has only one genuine signature). One of the access rights possessed by the resource which receives the message permits it to perform an operation which classifies the message as 'signed' or 'unsigned'.

Example 3 A class where only one qualifier is obligatory

This is provided by the example in section 4 of a resource which communicates with end users in order to authenticate them so that they may then access parts of the system for which they are authorised. It must have within its clearance at least the classes of the channels by means of which it is accessed by the end users and, therefore, of the end users themselves. One of its rights in respect of the class which includes the end users it authenticates is the ability to assign to messages it receives a particular subclass, corresponding to an end user.

Example 4 Registered 'Users'

A likely design for a secure system is that it records the clearance of each individual user and ensures that an end user, when authenticated, can access the parts of the system and the data for which he is cleared and nothing else. Using the model described here the information about such a registered user is a resource with a clearance which represents that of the user. The resource accessed initially by the end user (or some other resource or resources with which that in turn communicates according to the security rules of the system) is trusted to perform the correct mapping between the end user with his authenticated class and the class of the resource representing his use of the system. There is not necessarily a one to one mapping. On the one hand an end user may have more than one role to play in the system. On the other hand several end users may

perform the same work at different times, for example in a system where shift work is needed.

Example 5 Hierarchical Classifications

In military security a document is given a classification of 'unclassified', 'restricted', 'confidential', 'secret' etc. A person who may read such a document has a clearance, for example of 'confidential', which allows him, if there are no other restrictions, to read documents whose classification is equal to or less than his clearance (in this case the documents classified as 'confidential' or 'restricted' or 'unclassified'). Using the model described here this clearance is described by defining a class to represent it thus:

```
confidential-clearance : 1, confidential, restricted,
                         unclassified;
```

where 'confidential' etc. are attributes of the system. It is, of course possible to define classes that are not useful or which encourage insecurity, for example combining 'secret' with 'unclassified' and omitting the intervening ones. For this reason and for efficiency it is likely that the concept of hierarchies would be built into a practical system.

8 Security Policies

In terms of this model the security policy of a system is defined as follows. First define the total class of the system, with each of its subclasses. For each class thus defined state the classes of the system, with accompanying access rights, which may appear in the clearance of a component so classified.

This statement gives the maximum allowed clearance for any component. If a class is to classify only a passive component its maximum allowed clearance will be null. The total collection of maximum clearances for all classes is the security policy.

An active component may be created with less than its maximum allowed clearance in terms of classes, access rights or both. Since a component may not create another with a clearance more powerful than it possesses itself, this provides a way of creating locally security policies which are successively more stringent.

A security policy stated in this form is meaningful if the attribute used to define the classes of the system have a real meaning in terms of protection. Thus, for example, one might suppose that a resource class defined as '2, class-A1, secure-location' might be allowed a powerful clearance. There is a very real difficulty in pinning down the needed attributes and their meanings which is beyond the scope of this paper. Moreover, since the meanings given to the term 'security policy' vary and are not always well defined it would be rash to predict that the

definition given here will cover all of them. It is proposed as a tool for discussing security needs and consequent design.

A security policy can thus be represented diagrammatically as follows:

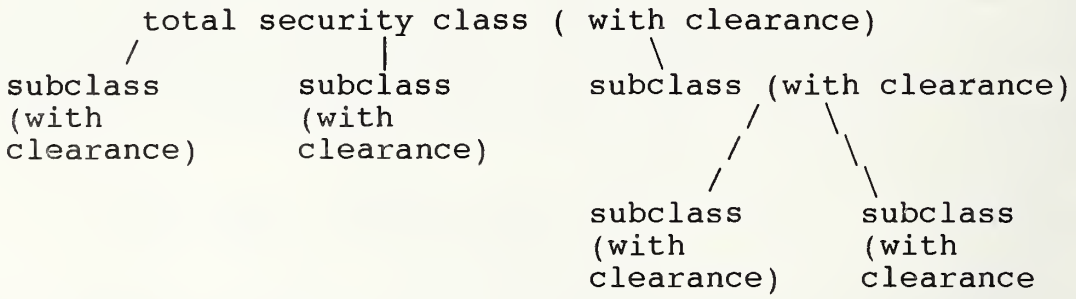


figure 7

9 Security Domains

Section 8 described a security policy in terms of the ideas of this paper. When discussing secure distributed systems the term 'security domain' is often used and is usually equated with a collection of communicating entities which are subject to the same security policy. In the terms of this paper a security domain is characterised by its security class and its security policy. This tells us what kind of domain it is. To identify it uniquely we need to identify its creator and, if necessary, the name given to it by its creator to distinguish it from others.

We may now consider the possibility of communication between entities which are in different domains. We may assume that there is a practical need to do this if we consider that two different commercial organisations may set up secure automated systems separately and may then develop the need to communicate.

Let us assume that, in each case, a total class for the secure domain has been defined, based on a set of attributes with a defined meaning in the environment to be automated. For each class and subclass access rights are defined which allow operations which are meaningful in the domain, either in terms of software available to the resources or operations performed by trusted personnel. For each class and subclass which can classify an active component the maximum allowed clearance is defined. Now, for interdomain communication a check must be performed that the clearance of the subject in one domain is, in some sense, compatible with the classification of the object in the other domain.

For the sake of simplicity let us assume that all communication between the two domains passes through a single resource, which will be called the interdomain gateway (or simply the gateway when that is unambiguous). In each of the domains the gateway has a classification which makes use of the attribute 'gateway'. Each resource which is allowed to communicate with a resource in the other domain has, as one of the subclasses in its clearance, the class which classifies the gateway and at least the access right which enables it to pass data to the gateway. Conversely the gateway has a clearance for each domain which enables it to access the resources which may take part in interdomain communication. This is illustrated in figure 8.

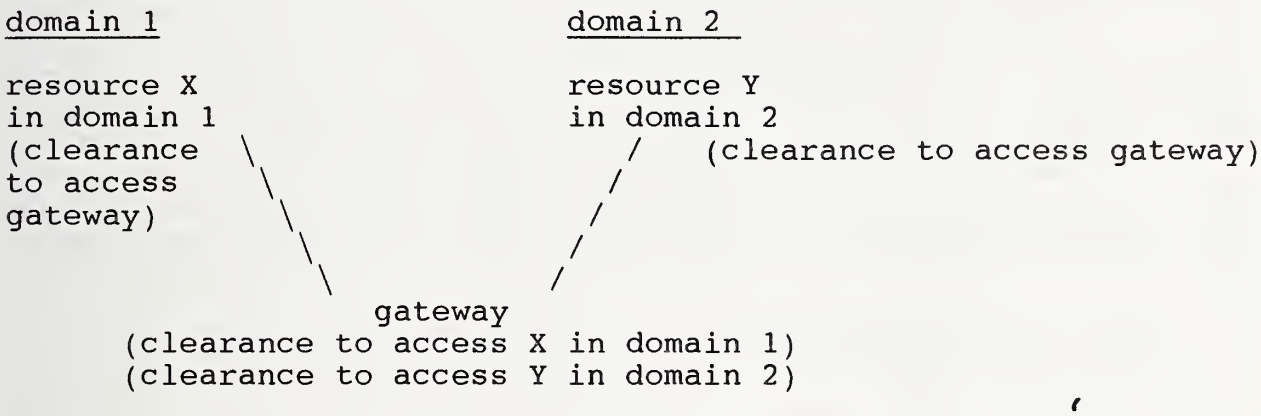


figure 8

The clearance of resource X to access other components is in terms of the classes of domain 1. To enable X to access components in domain 2 the gateway must contain information on the equivalence of classes and access rights in the two domains. The simplest arrangement is a one to one equivalence between a class/access right in domain 1 and a class/access right in domain 2. There need not be an equivalence in each case. Thus there may be classes in domain 1 which are inaccessible from domain 2 and vice versa. Similarly not all of the access rights may be made available to the other domain (e.g. information may be read from, but not changed in, the other domain).

It is conceivable that the gateway may need to recognise an equivalence between class/ access right combinations in the two domains where there is not a simple one to one relationship. This is beyond the scope of this paper.

Thus the gateway must hold a table of equivalences which it consults, together with the clearance of the would-be accessor, when access is attempted. The table is agreed and installed by collaboration of the management of the two domains. Secure domain construction and management is discussed in section 10.

10 Secure System Construction

Ref.1 described the secure construction of a system, starting with a completely trusted 'management entity' which had the power to create other entities and to devolve rights to them. It did not deal in classes (or therefore in classifications or clearances as described here). This section describes a similar process to that of ref.1, making use of these additional ideas. As a preliminary it considers what is gained by the additions.

In the description in ref.1 and here the starting 'management entity' which creates a distributed system is an organisation of people which is trusted to behave as a single entity to create those separate parts of the system which cannot be created by entirely automated means. The people perform procedures which correspond to the operations later performed by automatic entities to produce resources under their control and distribute rights among them. An elaboration of the concepts described in this paper is needed to describe in detail the operations to be performed both by the trusted people and by the automatic entities which they create. The comments made in this section therefore apply to procedures both by people and by automated resources.

The system of construction of ref. 1 provides the starting resource ('the management entity') with the ability to:

i) create other resources,

ii) provide them with rights, including the right to create other resources and to assign rights to them in their turn.

This produces a control hierarchy with which may be associated a complete record of which resources assign and use all rights in the system. However it lacks explicitly defined rules which state what rights may be assigned to particular resources. The introduction of the notion of a class enables a security policy to be defined in terms of the class and subclasses of the system being constructed and their access rights. When any resource is created it must be given a classification. Its maximum possible clearance is therefore defined by the security policy and an attempt to give it a clearance which does not accord with it is disallowed. In addition, each end user of the system has a classification and a clearance which derive from the classes of information he or she is allowed to receive from and send to the system and the operations to be allowed. Since an explicit classification is given to the resource which an end user may access a check is made when the end user receives access rights that his clearance matches the classification of the system he is allowed to access.

10.1 Creation of a Single Secure Domain

i) A total class is defined for the domain which is to be created. For the class and each of its subclasses a set of access rights is defined and for each access right a set of operations.

ii) A security policy is defined for the domain. It defines for each class /access right combination of the domain the attributes which must exist in the classification of the component which is cleared for that combination.

iii) The management entity for the new domain creates the resources which it is to control directly, assigning to each of them a classification and a clearance. In each case the compatibility of the classification and clearance is automatically checked against the security policy. In the case of a distributed system these directly controlled resources are those which exercise local control at individual locations. There are some resources (called basic resources in ref.1) which are controlled by the management entity and which exist before it has created any resources of the domain. These correspond to the hardware and software to be used and the newly created resources are given access rights to them as appropriate. Their clearance to receive these rights is checked automatically against the security policy. The classification of these basic resources is decided by the management entity and is part of the basic decision making on which the security of the system ultimately depends. The attributes used to classify the basic resources reflect the judgement of the management entity of the security features needed in and provided by the equipment for the real world environment in which the security domain is used.

iv) The directly controlled resources, thus created, create resources of their own and assign rights to their end users and to each other, as appropriate. These operations are automatically checked against the security policy.

10.2 Establishment of Secure Communications between Two Domains

Let us say that domains X and Y are to communicate according to mutually agreed rules of security. Then the following actions take place.

i) The management entities of X and Y agree a correspondence of those of their class/access right combinations which are to be used for intercommunication.

ii) The gateway between the two domains is formally created by both management entities and the table of correspondences is provided to it.

iii) Within each domain separately access rights are provided to the gateway and appropriate other resources so that interdomain communication can take place.

11. Practical Considerations and Conclusions

The model described in this paper has been developed bearing in mind the following principles.

i) Mechanisms which enforce security should be explicit in the system and separate from other functionality (e.g. from application code and system code which does not enforce security). There are a number of motives for this. It is more likely to be right, it is easier to change for those authorised, it may be made more difficult to change for those unauthorised, it is easier to check.

ii) The security enforcement mechanisms should be of as general an application as is necessary. This has affected the model in three ways:

a) the model is intended to apply to both military and civil applications and is therefore a superset of the features normally considered in relation to military systems;

b) it unifies some of the concepts of secure systems which have been elsewhere considered separately; authentication is treated as an authorised change of classification;

c) the model and the mechanisms which derive from it apply both to the construction and management of a secure system and to its subsequent use.

It is hoped to use the model both as the basis of the design of secure systems and as a means of relating security requirements to system design. The most immediate tasks are seen as a more rigorous description of the model and an assessment of its usefulness by comparing it with practical systems.

References

1. Jones R. W. 'The Design of Distributed Secure Logical Machines.', ICL Technical Journal, 1986 5(2).

2. Jones R. W. 'The Creation and Use of Explicit Rights in a Distributed System.', Proceedings of the Fourth IFIP Conference on Information Systems Security.

3. International Standard ISO 7498. Information Processing Systems - Open Systems Interconnection - Basic Reference Model.

4. Denning, D.E.R. 'Cryptography and Data Security'. Addison-Wesley, 1982.

5. Clark, D.D. and Wilson D.R. 'A Comparison of Commercial and Military Computer Security Policies.', IEEE Computer Security Conference, 1987.

6. ECMA/TC32-TG9/87/60. Security Framework for the Application Layer of Open Systems.

7. Department of Defense Trusted Computer System Evaluation Criteria. National Computer Security Center, July 1986.

8. International Standard ISO 7498/2. Security Architecture.

DBMS Integrity and Secrecy Controls

Rae K. Burns

Kanne Associates, Inc.
219 Bragg Hill Road
Ashburnham, MA 10430
(508) 874-5291

This paper addresses three areas that are being overlooked in the current discussions on data integrity:

1. A database management system (DBMS) provides the appropriate level of abstraction for the implementation of integrity controls as presented in the Clark and Wilson paper [CLARK87].

2. Application secrecy requirements and integrity requirements are interrelated; they must be analyzed and presented in a unified context.

3. The same DBMS security enforcement mechanisms apply to both secrecy and integrity.

1. Level of Abstraction

First, in order to discuss data integrity, one must define the domain of data that is of interest. The issues involved in, for instance, communications data integrity, are in fact different than those involved in the type of data integrity discussed in the Clark and Wilson paper. The Clark and Wilson concept of data integrity is based on two underlying principles, **well-formed transactions** and **separation of duty**. These principles lead to notions of **constrained data items** (CDI), **transformation procedures** (TP), and **integrity verification procedures** (IVP) for assuring overall system integrity. A distinction is made between certification procedures, which are performed manually, and enforcement procedures which are automated within the computer system.

The domain of data found in the Clark and Wilson concepts is precisely the domain that is addressed by database management systems. The Clark and Wilson notion of a constrained data item is not directly found in data communication systems or operating systems. It is, however, fundamental to database management. A database is composed of data items that are related to each other through a **database schema** that defines, for instance, records containing specific fields, tables or relations or files that contain specific records, and views

or subschema that define subsets of data items. A database management system provides numerous vehicles for defining an application's constrained data items. In addition, the notion of a well-formed transaction is also fundamental to database management. A DBMS transaction is a basic unit of database consistency and integrity; it is a well-understood concept that is integral to all database applications. Finally, the notion of integrity verification procedures maps to the database construct of **integrity constraints**. Integrity constraints are generally defined by the application but enforced by the DBMS. For instance, by specifying a primary key for a relation, an application can be assured that the DBMS will enforce the **entity integrity** rule for that relation.

It is clear that domain of applicability of the Clark and Wilson model is not an operating system or a network or even an application system, it is fundamentally a database management system. A DBMS provides the appropriate level of abstraction for implementing the enforcement mechanisms proposed in [CLARK87].

2. Secrecy and Integrity Interrelationships

One of the research results from the SeaView project was a clarification of the relationship between integrity and secrecy within the database environment. Within the SeaView security policy model ([DENN86],[DENN88]) there are several properties that relate integrity issues and secrecy issues. For instance, the **referential integrity property** [DENN88] requires, in effect, that all foreign key references "point downward" in classification level (e.g., an UNCLASSIFIED record cannot refer to a foreign key in a SECRET record). Additional SeaView security properties include the concepts of value constraints, classification constraints, and data correctness.

A fundamental relationship between classification constraints and integrity constraints is that a classification constraint is basically an integrity constraint that applies to a secrecy label attribute rather than a data value attribute. The problem of assuring that a set of such constraints is internally consistent and complete was discussed in [AKL87]. This is the same problem that has been a focus of significant database integrity research ([GARD79, [WILS80], [QAIN88]).

During the design of a database application, all aspects of the semantics of the application must be considered. To isolate secrecy and/or integrity semantics from the overall application semantics is fundamentally not feasible.

3. DBMS Enforcement Mechanisms

Within current database management systems, the discretionary access control mechanisms do not separate controls for operations that modify data (integrity) from controls that only read data (secrecy). For example, in a relational DBMS using SQL, the GRANT command includes the specific database operations of SELECT (read), INSERT, UPDATE, and DELETE (all of which modify the database). The same granularity of data access (e.g., database views) is used for both disclosure and modification operations.

If a DBMS were to enforce the transaction authorization controls described in the Clark and Wilson paper, they would be equally useful and effective for transactions that made no

modifications to the database. Their usefulness for disclosure, or secrecy controls, is based on the same separation of duty principle, and relates to what the military calls "need to know".

The concept of read access to data only by "well-formed transactions" defines the manner in which the information within a database may be combined and related. In other words, within many database applications, ad hoc queries are no more appropriate for disclosure than they are for modifications to the database. The user's role within the application organization defines which read-only transactions are appropriate as well as which update transactions. Within a multilevel database, for instance one that contains both SECRET data and TOP SECRET data, transaction authorization controls can reduce the potential TOP SECRET inferences a user might make based on solely on SECRET data. A user can be constrained to execute only "well-formed transactions" that have been "certified" to combine specific "constrained data items" only in a manner that does not facilitate the derivation of unauthorized inferences.

Summary

A database management system (DBMS) provides the appropriate level of abstraction for the implementation of application integrity and secrecy controls. The need for "well-formed transactions" for secrecy control is just as critical as for the control of data modifications. While the focus of the government concerns has been primarily disclosure, the need for secrecy and integrity controls is apparent in both commercial and military database applications.

REFERENCES

[AKL87] Akl, S.G., and Denning, D.E., "Checking Classification Constraints for Consistency and Completeness", Proceedings of the 1987 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1987, pp 196-201.

[CLARK87] Clark, D.D., and Wilson, D.R., "A Comparison of Commercial and Military Computer Security Policies", Proceedings of the 1987 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1987, pp 184-194.

[DENN86] Denning, D.E., Lunt, T.F., Neumann, P.G., Schell, R.R., Heckman, M., and Shockley, W.R., "Security Policy and Interpretation for a Class A1 Multilevel Secure Relational Database System", SRI International, 1986.

[DENN88] Denning, D.E., Lunt, T.F., Schell, R.R, Shockley, W.R., and Heckman, M., "The SeaView Security Model", Proceedings of the 1988 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1988, pp. 218-233.

[GARD79] Gardarin, G., and Melkanoff, M., "Proving Consistency of Database Transactions", Proceedings of the Fifth International Conference on Very Large Databases, October, 1979.

[QIAN88] Qian, X., "An Effective Method for Integrity Constraint Simplification", Proceedings of the Fourth International Conference on Data Engineering, IEEE Computer Society Press, February, 1988, pp. 338-345.

[WILS80] Wilson, G.A., "A Conceptual Model for Semantic Integrity Checking", Proceedings of the Fifth International Conference on Very Large Databases, October, 1980.

Work-In-Progress: Transformation Procedure (TP) Certification

Maria Pozzo
UCLA Computer Science Department

Steve Crocker
Trusted Information Systems

November 25, 1988

1 Introduction

Computer integrity is related to both an internal *and* an external consistency standard[2]. The external consistency aspect of integrity means that the data conforms to the real-world situation it is intended to describe[8]. Internal consistency is more concerned with the stability of the data, and the manner in which it is allowed to change. To preserve internal consistency, data should only be manipulated by well-formed transactions, i.e., data should never be manipulated arbitrarily by a user, but only in controlled and structured ways.

Clark and Wilson have proposed a model of data integrity that supports this definition[1]. A primary component of the Clark-Wilson Model is the Transformation Procedure (TP) which corresponds to the notion of a well-formed transaction. A TP is a program that has been certified to change the data it manipulates in constrained ways. This paper discusses a proposal for a TP certification mechanism based on formal specification and verification techniques. The next section provides an overview of the proposed solution. The general issues fall into three main classes: specification, analysis, and acceptance. These issues are discussed in detail in sections 3 and 4. Section 5 provides two examples of how this mechanism might work; each example is taken from a different domain. This proposal represents work-in-progress and the approach taken to study this proposal is discussed in section 6.

1.1 Components of the Clark-Wilson Model

Constrained Data Items (CDIs) are those data items within the system that come under the auspices of the integrity model. Unconstrained Data Items (UDIs) are objects whose integrity is *not* assured by the model. Two types of procedures can operate on CDIs: Integrity Verification Procedures (IVPs) and Transformation Procedures (TPs). The role of the IVP is to ensure that the data is consistent with its real-world counterpart at the time the IVP is run. This is analogous to an external audit of a bookkeeping system. The TP takes a CDI from one valid state to another valid state, thus preserving internal consistency as the CDI changes.

The model further defines five certification rules and four enforcement rules with respect to these entities. A complete list of these rules can be found in [1]. The rule that applies to the certification of TPs is rule C2:

> C2: All TPs must be certified to be valid. That is, they must take a CDI to a valid final state, given that it is in a valid state to begin with. For each TP, and each set of CDIs that it may manipulate, the security officer must specify a "relation" which defines that execution. A relation is thus of the form: (TPi, (CDIa, CDIb, CDIc, . . .)), where the list of CDIs defines a particular set of arguments for which the TP has been certified.

2 Overview

The goal of the mechanism proposed here is to certify a program as a valid TP. The general idea is to build a *filter* program which examines a potential TP to determine 1) if it manipulates only those CDIs that it is supposed to manipulate, and 2) if it does so in a manner that is consistent with the system's integrity policy for those CDIs.

The first part of this examination assumes the existence of some type of specification of what the program does. The term *specification* when applied to programs is usually taken to mean a general specification of all of the functional and/or other relevant properties of a program. Such a specification would imply the use of program correctness proof techniques, which have traditionally proven difficult to apply. Separation of the integrity issues from general program correctness simplifies the specification process. In the context of integrity, a primary concern is to restrict the data that a program can manipulate. In accordance with rule C2, this specification would be analogous to the "relation" that identifies the CDIs that a TP is

restricted to modify. To distinguish this form of specification from the more general use, the term *restriction* is used. The intent is that the filter then analyzes a program to determine if the program stays within its restriction. Although such an analysis is an inherently undecidable problem when viewed in a theoretical setting, the practical situation may not be so bleak. The specification and analysis issues are discussed in detail in the next section.

Once it has been determined that a given program is within its restriction, the next step is to decide whether the restriction permits actions that are acceptable according to the system's integrity policy. "The validity of a TP can be determined only by certifying it with respect to a specific integrity policy.[1]" For this reason, the integrity policy is seen as coupled with a CDI or set of CDIs. A program is certified as a TP if it implements the integrity policy associated with the CDIs it manipulates. These issues are addressed in more detail in section 4.

3 Specification and Analysis

3.1 Specification of the Restriction

The restriction is created by a program developer, system administrator or other security personnel, wishing to submit an executable program for certification as a TP. Each program must carry its own individual restriction which specifies which system entities the program may change. Syntactically, the restriction must be simple enough so that it can be machine processable by the filter program, and rich enough to represent the full range of modifications that occur in the real-world. The restriction must also be easily understood by the filter. This situation has analogy in labeling of packages containing food. Suppose someone is allergic to raisins and reads the ingredients list of every package before eating the contents to see if it contains raisins. If the manufacturer lists dried grapes as an ingredient, the consumer had better know that dried grapes are the same as raisins. Furthermore, the raisins may not be listed separately in any form. Some compound ingredient such as "Trail Mix" or "Swiss Power Fuel" may be listed. Hence the restriction has to be lucid and unambiguous if it is to be useful.

In order to gain some insight into the types of system entities that real-world programs modify, a small preliminary sampling of commonly-used programs was examined. The results from this sampling are not intended to provide any specific conclusions but to allow one to imagine what some

of the possibilities for a viable restriction might be.

List The restriction might simply be a fixed list of CDIs that are modified by the program. For some types of programs this might be a adequate.

Pattern Match Some programs such as compilers, editors, etc., create new files based on the arguments in the calling sequence or command line. For example, *spell* is a program that reads its input and creates a list of misspelled words in a file with a related name, e.g., "*spell* foo" creates the file "foo.sp". For such programs the restriction must identify the patterns of the names of the CDIs the program will modify. TPs such as editors, and compilers, often take a UDI as input and produce a CDI. Programs of this type may need additional certification according to rule C5[1], or the restriction could be expanded to include UDIs as well.

Complex rule The example above was based on transforming the argument list into a list of permissibly modifiable files using a simple extension of each filename. More complex transformations are possible and might be needed if the program computes filenames during execution. This is especially true of temporary files whose names are often created using some random combination of the program name and the date or time.

Type In a system where CDIs have a designated (or even hardware supported) *type*, the restriction might identify the type of CDI that is modified and the manner in which the modification is made.

The above list, although not exhaustive, identifies some of the ways that a specification could be constructed to restrict a program's modification activity to some set of system entities. In some cases, however, merely restricting a program to a set of CDIs is not enough. Consider a set of CDIs that contain unstructured Ascii data. In this case it is sufficient to state only which CDIs are modified; the manner in which the modification is made is not important. At the other extreme would be a set of highly-structured CDIs such as those that constitute a database. Here it is equally important to restrict the manner in which the modifications are made. An example of this would be a disk compactification program, which will appear to modify everything on the system and thus fail to meet any specification that is based solely on what CDIs are modified. For a program such as this, a formal specification must be stated in terms of semantic properties

to be preserved. In general, as the data contained in the CDI becomes more structured, the restriction must specify more semantic properties.

An added advantage of the restriction is its potential use as the "relation" which defines the execution of the TP. According to rule C2, the relation defines a set of CDIs for which the TP is certified. If the restriction cannot be used directly as this relation, it can be used to derive the relation which will be used to enforce the operations of the TP (rule E2 Clark-Wilson[1]).

3.2 Analysis

Once the language and format for the restriction is identified, the analysis issues must be addressed. How can a program be analyzed to determine if it is in conformance with its restriction? The filter must analyze the program and ensure that it modifies nothing except what the restriction specifies. The program being analyzed might generate code dynamically and then branch to it. Other problems arise with filename generation, as stated above, since the program might compute filenames in some very complicated way and then pass them to the operating system. An even more serious problem is that the program might make incisions into the operation system so that the operating system performs differently. Programs that are intended to operate as TPs should take full advantage of current software engineering techniques and need not contain these kinds of actions. For purposes of this paper, it is assumed that reasonably engineered programs will be analyzable. This is a somewhat large hypothesis and merits further discussion but is beyond the scope of this paper.

The analysis of even a simple program could prove to be a resource-intense operation. Since this is not a time-dependent activity, the analysis could be conducted on a machine separate from the target system. Once a program is verified to stay within its restriction, the program and the associated restriction must be "sealed" or encapsulated in some way to prevent tampering. One possibility is to encrypt the program and its restriction as a single unit. This would provide a trusted means for transferring the program and its restriction to the target system if, in fact, the analysis is conducted on a separate machine. Many issues arise when using cryptographic techniques for protection, particularly management of the encryption keys. These issues are well understood and will not be addressed here.

4 Acceptance

The process of accepting a program for installation on a particular system is independent of the analysis performed by the filter. If the program and its restriction have been properly sealed, it can be assumed that they have not been tampered with since they were verified by the filter. Given such a program and its associated restriction, the target system must first "break the seal" and then decide whether the actions that are specified in the restriction are acceptable according to the system's integrity policy for the CDI or set of CDIs that are manipulated by the program.

The DoD policy[4,5] which controls the dissemination of classified information requires protected objects to carry a marking or label which identifies the security level of the object. This security level implicitly defines how the object is handled with respect to the security policy being enforced.

Unlike the DoD policy which controls information dissemination, it seems unrealistic that a single integrity policy can be defined such that it is sufficient to handle a wide range of information and applications[7]. For this reason, an integrity policy is seen as application-specific, i.e., specific to the domain in which it is defined, perhaps coupled with the CDI or set of CDIs to which it applies. The difficult part of this is to define a policy that is broad enough to cover all the CDIs within the integrity perimeter of a particular domain and yet constrained enough that it can be implemented by the variety of TPs that manipulate those CDIs. The next section provides two examples from different domains and discusses issues concerning specification and acceptance.

5 Examples

5.1 Accounting Domain

This example is taken from the accounting domain. The potential TP is a program that posts journal entries to the appropriate ledger accounts in a double-entry bookkeeping system. The input to the program is a journal entry that identifies first the account to be debited and then the account to be credited, followed by the dollar amount[1]. A sample journal entry is shown in Table 1. Note that the dollar amount is the same for both accounts; this is because a credit purchase of equipment debits, or increases, the amount of

[1]Other information such as journal page number, date, explanation, typically found in real journal entries are peripheral to the discussion and omitted here.

equipment by $30,000 resulting in a credit, or increase, in accounts payable of $30,000.

Table 1: Sample Journal Entry

Page No.	Date	Description	Debit	Credit
14	*1-20-88*	*Equipment*	*$30,000*	
14	*1-20-88*	*Accounts Payable*		*$30,000*

The ledger accounts are highly-structured CDIs and the restriction must specify both the CDIs that will be modified and the manner in which they will be modified. The ledger accounts are assumed to have one of the following types: Asset, Draw, Dividend, Expense, Liability, Capital, Capital Stock, or Revenue. A *pseudo*-restriction[2] is shown in Table 2. The first item in the restriction identifies the program to which the restriction applies: *Post*. The next item in the restriction, Type, concerns the first argument in the input journal entry which is the account to be debited. In the example in table 1, *Arg1* is the Equipment ledger account which is an Asset account. The What item identifies the field in the account that is to be modified, in this case the *Balance*. The last item states the semantics of the change to the *Balance* field. If the ledger account has a type of Asset, Draw, Dividend, or Expense, a debit constitutes an increase in the balance by the dollar amount (recall that there is a single dollar amount). If the account has a type of Liability, Capital, Capital Stock, or Revenue, a debit constitutes a decrease in the balance by the dollar amount. In this example, a debit to Equipment, an Asset account, results in an increase in the balance by $30,000. The restriction is repeated for *Arg2* which is the account to be credited.

Note that the restriction does not make any statements about the algorithm. It simply states what is modified and how it is modified: increased or decreased depending on the type of ledger account and whether it is a debit or a credit. It is assumed that other tests are performed to determine the correctness of the algorithm, in this case a simple increment or decrement of the appropriate account balance.

[2]The format of the restriction in Table 2 is just an example and may bear no resemblance to the final format of a restriction when this work is complete.

Table 2: A Possible Program Restriction

Program:	*Post*
Type:	*Arg1*
What:	*Balance Field*
Semantics:	
(1)	$Arg1 = (Asset, Draw, Dividend, Expense) \Rightarrow \uparrow Balance(\$)$
(2)	$Arg1 = (Liability, Capital, Capital\ Stock, Revenue) \Rightarrow \downarrow Balance(\$)$
Type:	*Arg2*
What:	*Balance Field*
Semantics:	
(1)	$Arg2 = (Asset, Draw, Dividend, Expense) \Rightarrow \downarrow Balance(\$)$
(2)	$Arg2 = (Liability, Capital, Capital\ Stock, Revenue) \Rightarrow \uparrow Balance(\$)$

Given the restriction and the potential TP, the analysis phase of the filter must verify that the program actually changes only those CDIs as identified in the restriction and only in the manner identified. Once the program has been shown to stay within its restriction, the next step is to check the integrity policy for the CDIs that will be modified. The integrity policy for the ledger accounts is relatively simple. It states that posting to ledger accounts occurs in pairs: a debit and a credit; and the dollar amount is the same for both accounts. In the example given, the *Post* program implements this integrity policy for the ledger accounts. Further, if the accounts begin in a valid state which in this case is $Assets = Liabilities + Capital$, then any TP which implements the policy will preserve this property and will take the CDIs (ledger accounts) to a valid final state. Note that an Integrity Verification Procedure (IVP)[3] is still needed to ensure that the accounts represent the true state of the real-world entities they correspond to.

5.2 Computer Viruses

One of the biggest integrity threats to general purpose systems is the computer virus[3]. A computer virus is a program that contains some hidden function that could have harmful side-effects. Recently, viruses have been attracting substantial attention because they propagate and can affect a

[3]The role of the IVP, as defined by Clark-Wilson, is to ensure that the data is consistent with its real-world counterpart at the time the IVP is run.

large number of people and systems. The potential is no longer theoretical. In late 1987, viruses began to attack the PC community destroying user files, flooding major networks, modifying boot programs, and a host of other malicious tasks resulting in the loss of thousands of dollars of information, time and resources [6]. Most recently, a virus has attacked a many systems on a major world-wide network.

A computer virus causes an integrity violation because it spreads throughout a system by modifying other executable programs to include a copy (or a mutated copy) of itself. In this domain, the CDIs are the executable programs to be protected from unauthorized modification. Executable programs that are load modules generally contain some structured header information followed by the unstructured executable portion. Executable programs that are interpreter input files can be thought of as unstructured Ascii data. In either case, the issue is not *how* these programs are modified, but what programs are allowed to modify them. The important issue is that only certain programs are authorized to modify executable programs. Thus, a program that is potentially a TP, must have a restriction that identifies the CDIs it modifies. This will result in a restriction that is simpler than the one proposed for structured CDIs in the accounting example because it will not contain any semantic properties.

Although specifying the restrictions may be simpler in this domain, defining an integrity policy, or anti-viral policy, is more difficult. One reason for this is that it is necessary to distinguish between ordinary modifications and viral modifications; a problem generally considered undecidable. For example, suppose the anti-viral policy states that only a trusted compiler is allowed to perform modification operations on executable programs. A potential TP that is a run-time debugger would have a restriction that includes executable programs in the list of CDIs that it manipulates. Such a debugger would not be accepted since it does not implement the system's anti-viral policy regarding executable programs. Furthermore, any potential TP in this domain that is not a trusted compiler would have to exclude executable programs from the list of CDIs that are manipulated in order to be accepted by the filter. Thus, viral spread may be greatly reduced or even eliminated but the trade-off is the loss of some potentially useful programs such as the run-time debugger.

The proposal for such a filter is being investigated from this domain. The details of the approach being taken to determine if a virus filter is feasible is described in the next section.

6 Approach

In the case of a virus filter, the restriction states what the potential TP modifies. It is believed that the vast majority of programs have relatively simple restrictions, unlike the disk compactification program described earlier, and that it will be straightforward to certify most useful programs as TPs. This leads to the following hypothesis which is central to the virus filter proposal:

Hypothesis 1 *Is it possible to formulate restrictions for the majority of useful programs such that the restriction is syntactically simple enough to be machine processable and fine-grained enough to represent the full range of authorized modifications made by real programs?*

One way to proceed on this hypothesis is to increase the sample size to a larger set of programs used in some environment, e.g., a hundred or so programs commonly used in the DOS environment, and attempt to write the restriction for each one. Programs in the sample which appear to have very complex specifications such as the disk compactification program will be negative examples with respect to Hypothesis 1. If there are only a few such programs, then other techniques can be used to examine those programs and Hypothesis 1 will hold.

The variable of interest in the real-world analysis is what type of entities does a program modify and what are the characteristics of the modification. For example, create vs. append, global variable vs. parameter, fixed file name vs. randomly generated name. The question to be answered is: can restrictions be written for a majority of programs in compliance with Hypothesis 1? Obviously, there is no way to identify if restrictions can be written for *all* programs just as there is no way to cover *all* types of modifications. For these reasons, the initial real-world analysis will be restricted to a sampling of programs found in one or two environments which are representative of the types of systems that are currently being used extensively, e.g., an MS/DOS environment and a Unix environment.

Another important factor in such an analysis is the sample size which must be large enough from which to draw legitimate conclusions, and small enough to be manageable. Initially, several hundred programs will be analyzed distributing the selection among multiple environments being sure to consider a mix of both commercially marketed software and free software (shareware). Also important is the criteria used to select the "commonly used" programs. In general, this will be done by selecting programs that

users most frequently use in the environment(s) being considered, as well as the authors' knowledge of users and systems. Lastly, the analysis itself will involve source code examination, documentation review, conversations with program developers, and perhaps some program testing but will not attempt to use any formal method.

Because the restriction is much simpler in this case, it is believed that most useful programs will be analyzable. This leads to the second hypothesis which is also at the heart of the virus filter proposal:

Hypothesis 2 *Is it possible, on the average, to analyze programs in a straightforward way?*

To test this hypothesis, the virus filter program will be developed in stages. Each stage will add a deeper analysis than the last. Hypothesis 2 will be true if relatively few stages are needed to yield a filter that correctly verifies programs according to their restriction.

For the first stage of filter development, a formal description of the instruction set of the target computer would be prepared. For example, if the target computer was built on an Intel 8088, a formal description would be prepared to include the entire instruction set of the 8088 plus the interactions with I/O such as the disk, screen and keyboard. Each instruction type would be characterized by the locations that are modified, e.g., registers, memory or I/O, as well as the instruction flow control – next sequential instruction, branch, jump or trap. This characterization needs to be complete. For example, it is not sufficient to state that a store instruction modifies memory, it is also critical to identify what location in memory is changed.

Using the formal description of how the instructions work, a map of the executable portions of the address space versus the data areas must be built for each program to be examined. Together, the formal description of the instruction set and the program's map can determine the portions of the address space that are subject to modification. Next, all store operations and all calls to the operating system will be examined to identify the files that are going to be modified. These questions are likely to be hard since most programs contain indexed addresses and pointers, and it is possible that such indirect modifications will be left to subsequent stages of filter development. However, it is possible that bounds can be identified on what the effective addresses are using relatively simple analysis techniques drawn jointly from the fields of compiler optimization and verification. Similarly, some tracing or symbolic execution may be needed to determine the filenames that are being passed to the operating system in a system call, but it is anticipated

that most benign programs do relatively little transformation of filenames. Using this information regarding the modifications that the binary actually makes, the comparison can be made with the restriction. Programs whose modification operations are not confined to those specified in their restriction are rejected at this point.

Once the analysis is complete and the filter has verified that a given program is within its restriction, the next step is to decide whether the actions that are specified are acceptable, i.e., according to the target system's policy regarding viral activity. As previously stated, this is a difficult problem and involves the ability to distinguish between ordinary modifications and viral modifications. It may be possible, however, to classify modifications such that the loss of useful programs, such as the run-time debugger in the previous example, is minimal. The third hypothesis concerns anti-viral policy as follows:

Hypothesis 3 *Is it possible to classify modifications such that ordinary changes can be distinguished from suspicious ones and the loss of useful programs is minimized?*

To test Hypothesis 3, the suite of programs from the real-world analysis, augmented by programs known to contain viruses, can be used to categorize the types of modifications being performed. Clearly, modifications that change the program itself, or tamper with the operating system would be considered suspicious. Hypothesis 3 will be true if a large percentage of modifications performed by programs in the test suite can be classified in such a way so that few useful programs are rejected, while *ALL* programs which actually contain viruses are rejected.

Once the acceptance phase is added to the filter, the entire mechanism must be tested. The programs in the sample can be used as the test suite. A value of "ok" or "not clearly ok" will be returned for each program. For each program that is rejected, some indication of why the program was rejected will be provided. Rejected programs will be analyzed by hand in more detail. Three results are possible:

1. The system's anti-viral policy is clearly violated – a virus is detected. In this case the filter has correctly rejected a program that the system considers to contain a virus.

2. The program does not violate the system's anti-virus policy but it has coding practices that are beyond the analysis capability of the filter. Further analysis techniques may have to be added to the filter.

3. The filter exceeds some tolerable time bound for analyzing the program. In this case the program is too complex to analyze and the addition of stronger, and presumably more time-consuming analysis techniques, is not acceptable.

Based on the list of programs which are falsely rejected and a review of the reasons for these false rejections, it will be possible to determine whether this approach to filtering viruses is feasible.

7 Conclusions

7.1 Summary

An approach for certifying TPs has been proposed that is based on formal specification and verification. Two examples have been presented: one in an accounting domain where the CDIs are highly structured data items; and a second from the viewpoint of computer virus protection where the CDIs are generally unstructured entities. This proposal includes the specification of a program's modification activity, called a *restriction.* It has been shown that as the system entities to be protected become more structured, the specification becomes more complex and may also include semantic properties as well as the identification of the CDIs that are manipulated.

From the two examples shown, it appears that as the restriction becomes more complex, the integrity policy is simplified. Although this is true for the two examples described in this paper, it may not be true in general. Nonetheless, an integrity policy is seen as application-specific, i.e., specific to the domain in which it is defined, perhaps coupled with the CDI or set of CDIs to which it applies. At the very least, an integrity policy must be broad enough to cover all the CDIs within the integrity perimeter of a particular domain and yet constrained enough so that it can be implemented by the variety of TPs that manipulate those CDIs. It seems unrealistic that a single integrity policy can be defined for *all* domains.

7.2 Work In Progress

Currently, work is underway investigating the feasibility of applying the filter approach to virus protection. The success of a virus filter approach rests on whether the following three questions all have affirmative answers.

1. Is it possible to formulate restrictions for the majority of useful programs such that the restriction is syntactically simple enough to be ma-

chine processable and fine-grained enough to represent the full range of authorized modifications made by real programs?

2. Is it possible, on the average, to analyze benign programs in a straightforward way?

3. Is it possible to classify modifications such that ordinary changes can be distinguished from suspicious ones and the loss of useful programs is minimized?

To answer these questions, experimental work is required to establish whether a significant fraction of the useful programs are subject to formal specification and analysis of the form envisioned here. Once the experimental work is done, it will be clear whether this approach is feasible within the current state of the art, requires modest extension to the state of the art, or is likely to remain beyond the state of the art. If this approach is feasible for filtering viruses, the proposal can be extended to other domains.

References

[1] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Security Policies. In *IEEE Symposium on Security and Privacy*, pages 184–194, April 1987.

[2] David D. Clark and David R. Wilson. Evolution of a Model for Computer Integrity. In *Proceedings of the 11th National Computer Security Conference*, October 1988.

[3] Cohen, F. Computer Viruses. In *Proceedings of the 7th National Computer Security Conference*, pages 240–263, 1984.

[4] Department of Defense. *Security Requirements for Automatic Data Processing (ADP) Systems*, April 1978. DOD Directive 5200.28.

[5] Department of Defense. *ADP Security Manual – Techniques and Procedures for Implementing, Deactivating, Testing, and Evaluating Secure Resource-Sharing ADP Systems*, June 1979. DOD 5200.28-M.

[6] Highland, H.J.,. Random Bits & Bytes. *IFIP Computers and Security*, 7(2), 1988.

[7] Robert H. Courtney Jr. Integrity Workshop. Dockmaster Forum, January 1988. Conversation between Robert Courtney and Stu Katzke concerning the Oct. 1987 WIPCIS Workshop.

[8] William H. Murray. Position Paper: Working Group on Granularity. In *Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)*, October 1987.

Toward a Model for Commercial Access Control

Stan Kurzban
IBM Systems Research Education Center
500 Columbus Avenue
Thornwood, New York 10594

ABSTRACT: Access control software has developed to meet commercial requirements, but no discipline has made it possible to describe the software rigorously. This paper presents a set of objects that are suitable for use in rigorous descriptions of such software.

Introduction

For fifteen years, access control software has been developing to meet commercial needs. Software designers have introduced a diverse collection of mechanisms and facilities in response to imprecisely phrased requests for types of control. Making sense of the set of software offerings requires a uniform and precise set of terms with the power to express all of the relevant things that the software does. Relevance, in this context, depends on perception of commercial needs.

Origins of Requirements

A long history of accounting practice indicates that these needs arise from Generally Accepted Standards of Good Practice (GASGPs). Most important of these are:

1. Least Privilege (LP)--People must be authorized to do all and only what they must do to perform their assigned tasks.
2. Separation of Duties (SD)--If a set of acts can jeopardize the organization, then multiple individuals with potentially conflicting motives must be involved in performing the set of acts.

The first of these GASGPs insures that the organization will not incur unnecessary risk by permitting employees to do things that they need not do to serve the organization's interests. The second insures that collusion or one person's duping of another will be necessary to the commission of a dishonest act that injures the organization. History has amply taught that people who do unnecessary acts often do them incorrectly and detrimentally and that individuals who collude to do harm are often caught. History also makes it clear that security measures that are difficult to use are omitted. That lesson leads to another fundamental principle:

- Ease of Safe Use (ESU)--The easiest way to do something should always be the safest as well. This entails, for example, the use of safe defaults, that is, specifications such that the failure to enter a value for a variable results in the variable's assumption of a value that confers least privilege. It also entails the provision of grouping mechanisms that make it possible for people to enter security-related data with a minimum of commands/keystrokes. The use of list-based access control rather than resource-related passwords is still another example.

Finally, all is for naught if measures are ineffective against attack. Therefore, a *sine qua non* is adequate invulnerability:

- Adequate Invulnerability (I)--Security measures must be implemented so well that deliberate circumvention of them will be demonstrably less attractive than other means of achieving the same objectives (for example, in a particular case, system penetration by sophisticated programmed attack will be less attractive an alternative than duping or colluding with an authorized individual). While the definition of invulnerability must be precise, specifying what is permitted and what is prevented, the confidence that one can have in invulnerability is subjectively quantifiable.

First Consequences

Clark and Wilson (1) used the principles mentioned above in deriving a "model" for commercial computer security. Their work and our own suggest a few fundamentals of commercial access control:

1. Because applications rather than systems provide the level of granularity needed for effective limitation of privilege (LP), the facilities of access control software must be available to application software.
2. Because administrative ease (ESU) requires abbreviation of administrative statements, access control software must permit the aggregation of both users and resources into groups. (Thus, a statement about a group of twenty users and a set of fifty resources is an abbreviated form of a thousand statements, one for each ordered user-resource pair.)
3. Users must be able to prove their identities easily (ESU), as well as with certainty (I).
4. The separation of administrative tasks by scope (for example, for a set of groups of users), called "decentralization," and responsibility (SD) (for example, auditing or security administration) is vital, to permit the placement of only tolerably small burdens on administrators (ESU).

Because many people now use diverse systems, Ease of Safe Use requires that they not be confronted by unnecessarily differing access control interfaces. The users concerned are developers of applications, those who use applications, and administrators. Their needs are:

1. Application Developers:
    a. The ability to learn whether the application's current user is authorized to make the requested type of access to the named application-defined resource and to log the learned information.
    b. The ability to create, associate with a list of those authorized the various types of access, modify, and destroy collections of access control data, called "profiles," with application-defined resources.
2. Application Users:
    a. The ability to initiate a series of interactions with the system (that is, to "LOGON"), prove their identities, and associate their work with one of possibly many groups to which they belong. "System," here, refers to all that a user sees during a series of interactions, whether that be a single application, a single processor, a network, or even many interconnected networks.
    b. The ability to create, associate with a list of those authorized the various types of access, modify, and destroy collections of access control data, called "profiles," with resources.
3. Administrators:
    a. The ability to modify profiles.
    b. The ability to modify the the constituency and semantics of groups of users and resources.
    c. The ability to modify the security-related data associated with users and with the system as a whole.

A vital inference that Clark and Wilson derived is that control of access to application-defined data requires that all control of the data be exercised by the application. They focus on user-program-data triplets; every privilege has the form: USER may gain access to DATA via PROGRAM. A simpler interpretation of their view uses PROGRAM as a type of access. In this formulation, we never say CHRIS may READ EMPLOYEE or CHRIS may READ EMPLOYEE via PAYROLL, but rather, CHRIS may use PAYROLL on EMPLOYEE. The program's name replaces the type of access in a conventional access control triple. The device is most appealing, but herein, we retain the explicit notion of program as intermediary.

The Model's Objects

The above indicates that the objects of interest for commercial access control are:

1. User Profiles, which comprise security-related information about users.
2. Group Profiles, which we rename "role-surrogate" (RS) profiles to indicate that in commercial data processing, as Lee (2) has noted, useful grouping depends on users' organizational roles. Since these reflect the structure of the organization, which is typically hierarchical, we assume that a net of these must be representable.
3. Resource (and Resource Set) Profiles, which comprise security-related information about (sets of) resources.
4. Programs, which may be types or conditions of access permitted.
5. Processes, which are the units from which requests for access emanate.
6. System-wide Security Data, which comprises at least data on the semantics of RS profiles.

The last of these tells the access control software how to use the other information in answering the question of whether access is permitted.

Below are listed, in the context of our model, the contents of the items in the above list. Items marked with stars are not implied by the preceding discussion, but are included because they commonly appear in access control facilities and we wish to show how they might logically be grouped for convenience. The notion of "contents" is used loosely. It can be thought of in terms of the fields of a control block or the values to which the fields point. If a model is developed from this description, the N contents listed below may correspond to an N-tuple that represents the object in the model.

A User Profile contains:

1. The user's system-local and *network-wide or *global identifiers.
2. Data used to verify the user's identity.
3. The identifiers of the RSs to which the user belongs. (Note: This duplicates information in the individual RS profiles and, in practice, need not be stored redundantly. It is shown so that one may make statements about "all the information in the User Profile" and include this item, even though the information is actually stored elsewhere. The point is that such a statement implies that the access control software must retrieve the data, from wherever they are stored, in the process of conforming to such a statement. The same principle applies to all data shown redundantly below.)
4. For each granting of access to the resource (set):
   a. The identifier of the resource (set),
   b. The type of access granted, and
   c. The conditions (for example, *times of day, day(s) of week, *port of entry, program in clean (that is, meeting the System Integrity criteria (3) of IBM's Multiple Virtual Storage (MVS) operating system) calling process).

   The presence of this item permits the model to describe a ticket-based access control mechanism (as contrasted with one that is list-based). (Note: This duplicates information in the individual resource (set) profiles and, in practice, need not be stored redundantly. It is shown because the information may be stored in this way (for example, via capabilities, as in a capability-based system like AS/400 (4)) and so that one may make statements about "all the information in the User Profile" and include this item, even though the information is actually stored elsewhere. The point is that such a statement implies that the access control software must retrieve the data, from wherever they are stored, in the process of conforming to such a statement.)

5. Identifiers of all resources and types of access thereto for which the user has GRANT authority (per Bishop (5)) (that is, those whose profiles' access lists the user may modify and with respect to which types of access).
6. Audit-control data. (Detail not given here.)
7. *The user's United States Mandatory Access Control (MAC) (Bell and La Padula (6)) clearances.
8. *Conditions under which the user may not be permitted to use the system (for example, outside of normal working hours, via telephonic communication lines, on the weekend, on certain dates).
9. *Default resource profile for newly created objects.

An RS profile contains:

1. The RS's identifier.
2. The identifiers of the users who belong to the RS.
3. The identifiers of the RSs immediately above and below this one in the net of RSs.
4. The identifiers of the resource (set)s for which this RS has GRANT authority.
5. Audit-control data. (Detail not given here.)
6. The scope of the RS (that is, identifiers of the profiles and the types of contents thereof to which this RS applies; for example, if the scope of this RS is the RS "Payroll" and the field "Audit-control data" and this RS has access of type UPDATE to the resource set of all profiles, then making people members of this RS is the same as making them Group Auditors for the PAYROLL group, in the terminology of IBM's Resource Access Control Facility (RACF) (7)).
7. *RS-related privilege assigned by default to members of this RS.

A resource (set) profile contains:

1. The identifier of the resource (set) and the type of the resource(s). The type is used as an index into a table of types to determine the bit-correspondences for specified types of access to resources.
2. Either:
   - If a single resource, the identifiers of all resource sets to which this resource belongs, or
   - If a set, the identifiers of all the resources that belong to this set.
3. Identifier of the owning RS or user.
4. For each granting of access to the resource (set):
   a. The identifier of the user or RS,
   b. The type of access granted, and
   c. The conditions (for example, *times of day, day(s) of week, *port of entry, program in clean (that is, meeting MVS's System Integrity criteria (3)) calling process).

   The presence of this item permits the model to describe a list-based access control mechanism (as contrasted with one that is ticket-based).
5. *The resource's United States Mandatory Access Control (MAC) (Bell and La Padula (6)) sensitivity level and categories.

Associated with each program are:

1. Its identifier.
2. The identifiers of all resources for which it is the condition of access. (Note: This duplicates information in individual resource (set) profiles and, in practice, need not be stored redundantly. It is shown to account for the case (for example, AS-400 (4)) where the information is logically (albeit not quite physically) stored in this way and so that one can make statements

about "all information associated with the program" and include this item even though the information is actually stored elsewhere. The point is that such a statement implies that the access control software must retrieve the data, from wherever they are stored, in the process of conforming to such a statement.)

Associated with each process are:

1. Its identifier.
2. The identifier of the user on whose behalf the request for access is being made.
3. The identifier of the RS associated with that user.
4. The identifier(s) of the currently active program(s).
5. The "clean" bit (see above).
6. *MAC.

System-wide security data are:

1. The rules of search for determining whether access is permitted. This is expressed as an ordered list of search index and the role of each. Each entry in the list is composed of:
   a. A search index, which may be:
      1) An identified user or RS profile (for example, PUBLIC or UACC) to be sought,
      2) The current connect RS, or
      3) Other RSs of which the user is a member; and
      4) RSs below or above those of which the user is a member.
   b. A role:
      1) An override (that is, if some type(s) of access is (are) specified, use it if (in)sufficient and search no more), or
      2) A default (that is, if some type(s) of access is (are) specified, use it if (in)sufficient unless a more permissive (set of) type(s) is found later.

   Note that mechanisms for efficient implementation, like the Global Access Control (GAC) table in RACF(7) and rules about ownership (that is, "resource's name begins with user's identifier") are not distinguished in the model, but are represented by the appropriate information in the appropriate access lists. It is as though the owner's name appears with total authority in the access list for each owned object.

   Note too that detailed presentation of all of RACF's (7) and OS/400's (6) searching rules is not given here, but the things from which that presentation can be built are, we hope, given.

   Finally, note as well that some of the apparent redundancy described above may gain semantics from a search order. That is, if a piece of information such as "LES may READ X" is stored multiply, but in places that are separated by one or more others in a search, the semantics of the multiple instances may differ because of the role that each plays in a search. This is another problem whose resolution is "not given here."
2. Audit data that have been recorded. (Detail not given here.)
3. *Security switches (for example, the default for the setting of the "erase-on-scratch" feature)
4. *Password control rules (for example, every password must contain at least one digit, passwords must be changed every thirty days).

Further Work

The natural language exposition above is only a step on the road to a formal model. Clearly, a vital next step is mapping of the items described to some formalism that would permit the development

of a model that the "Discussion" below posits. Whether that step would be unduly difficult is a matter of conjecture that only its execution can settle.

Discussion

The model, after formalization, refinement, and extension, could serve multiple purposes:

1. An agency could establish criteria, portions of the model that a product would have to implement to be awarded a rating of some sort.
2. Architects could establish a set of model-defined facilities that would be an architecture for access control.
3. Verifiers could test an access control product against its vendor's model-based statement of function to determine whether it functions as claimed.
4. Such verification could be automated via theorem-provers and the like.

Conclusion

While forces in the marketplace have led to the development of access control software that appears quite diverse, certain primitives, formularized in the model described above, underlie almost all of the functions they perform. These primitives provide an opportunity for treating their similarities and differences rigorously, albeit with much effort.

References

(1) Clark, D. D., and Wilson, D. R., "A Comparison of Commercial and Military Computer Security Policies," *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, April 27-29, 1987, Oakland, California, pp. 184-194.

(2) Lee, T. M. P., "Using Mandatory Integrity to Enforce 'Commercial' Security," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, April 18-21, 1987, Oakland, California, pp. 140-146.

(3) International Business Machines Corporation, *Statement of MVS System Integrity* P73-17, February 1, 1973.

(4) International Business Machines Corporation, *AS/400* (TM) *Programming: Security Concepts and Planning*, SC21-8083, June 1988.

(5) Bishop, M., and Snyder, L., "The Transfer of Information and Authority in a Protection System," *Proceedings of the Seventh Symposium on Operating System Principles*, Pacific Grove, California, December 10-12, 1979, pp.45-54.

(6) Bell, D. E., and La Padula, L. J., *Secure Computer Systems: Unified Exposition and Multics Interpretation*, MTR-2997, The MITRE Corporation, Bedford, Massachusetts, July 1975. (ESD-TR-75-306)

(7) International Business Machines Corporation, *Resource Access Control Facility (RACF) General Information Manual* GC28-0722, May 1987.

APPENDIX B

The Position Papers

Data Integrity Position Statement

Marshall D. Abrams
The MITRE Corporation
McLean, VA 22102

I think that most of the discussion of integrity has concentrated on describing functionality. This is important work and needs to be continued. But it is not sufficient. In order for people and organizations to be able to rely on, or trust, the integrity of their computer systems and networks, they must be convinced that that the integrity policy is implemented correctly, that it is error free, and sufficiently resistant to accidental or purposeful circumvention.

This trust in integrity mechanisms is equivalent to trust in secrecy mechanisms. The process of establishing the trust is commonly known as assurance.

I think that progress concerning integrity has, so far, concentrated on functionality. I think that commensurate attention must be given to assurance.

Process Execution Controls as a Method of Ensuring Integrity

DRAFT

Eugen Mate Bacic

Software Security Specialist
Communications Security Establishment

NB: This is a draft of a larger paper discussing a proposed method of ensuring the integrity of a system. It may ramble at times and for this I apologize. Hopefully the followup paper will correct these oversights.

. Introduction

Integrity is defined "... as those qualities which give data and systems both internal consistency and a good correspondence to real-world expectations for the systems and data" [Clark & Wilso . This implies that the "...system remain predictably constant and change only in highly controlled and structured ways". What follows is a method of ensuring this structured and controlled rate of change remain structured and controlled by means of finite state machine. Any change within the system would be forced to move from one known steady state to another. Much as a compiler uses a finite state machine to move from one legal state to another, the method proposed allows the operating system to ensure that no matter what the user does with the system, it moves from one known state to another known state.

Process execution controls (PECs) are offered as a method of ensuring the integrity of the system by limiting the accessibility of a wide range of data to modification. All data on the system would be governed by various execution controls which would, from the viewpoint of the program, restrict who can execute which program. PECs would behave similarly to mandatory access controls (MACs) and discretionary access controls (DACs) [DoD 85] but instead of applying the normal accesses of a user to a file or the classification levels of data, PECs would apply a set of controls to data restricting what programs or individuals can access them. This restriction on usage by program or individual has become known as MEC and DEC, paralleling MAC and DAC.

2. Process Execution Controls

Mandatory execution controls (MECs) are truly an extension of access control lists (ACLs). However, their mandatory nature implies that they can not be changed, unlike ACLs. At the discretion of the user would be discretionary execution controls (DECs) which could be modified to include certain users or groups of users in ways meaningful to the owner of the data.

Vtsually represented, PECs could be viewed as requiring an additional step that MAC and DAC do not require. This extra step, in which an object becomes the subject for the next object, increases the security and nicely falls into line with the Clark-Wilson model:

```
1.  Subject        Object
2.                 Subject              Object


     User ===>      Program      ===>    Data
  (Duties)     (Req'd Duties)           {MAC/DAC}
                                        {MEC/DEC}
```

This diagram shows how a User, with a given set of duties, executes a Program, with a given set of required duties, which then executes on behalf of the User and attempts access to the Data. As is indicated, firstly the User is the subject and the Program the data. After the Program takes over for the User, the Program becomes the subject and the Data becomes the object. Although one might indicate that this could collapse down to only one subject and object, ever, it will be shown that the above method is required to fully utilize PECs thus ensuring integrity.

2.1 Duties

For this method to work requires that we define what is meant by duties. Duties are a set of processes for which the given user has access to. In other words, every given duty, from a simple user to system administrator, has a set of processes associated with it. In the case of a developer user these duties might simply define which processes he can execute:

i.e., loader, link, editor, debugger, cc, pascal

while for a business user might have:

i.e., loader, editor, payroll, cheques-balances.

Now, although this might seem clumsy at first, it can quickly be simplified by using groups and having the loader, link, editor, ... all listed under a group name: Developer.

Therefore, a user whose duty was Developer would be able to use the loader, linker, editor, debugger, c compiler and pascal compiler. However, a user who's duty is Business-Payroll would not have access to the linker, debugger, c or pascal compilers but would have access to the payroll and cheques-balances programs. This description of duties down to the lowest common denominators -- processes -- is required for the PEC system to work.

2.2 Mandatory Execution Controls

Mandatory execution controls (MECs) are those controls imposed upon data in the system at the time of creation -- be it creation of the system or creation of a data entity. MECs, being mandatory, appear within all files. They can not be modified nor added to. They dominate discretionary execution controls (DECs) but work in conjunction with MAC and DAC. If MAC and DAC disallow execution, neither MEC nor DEC can supersede it.

MECs take the form of process names. Every file is considered to be an object which can be manipulated by some subject. Some objects -- commonly called executables -- can become subjects on behalf of their executor (at one time a user) and manipulate objects. It is upon the objects that the MECs are placed. Whenever a file is to be accessed -- even execution is a file access -- the MEC list is examined. If the current subject's duties is contained within the MEC list of the object, the object is opened by the calling subject. From this open, the MAC and DAC lists apply. If the file is not accessible, the open will fail. If the file is accessible, but an open is attempted in the wrong mode, the open will fail. With this we can ensure that data is not overwritten or modified by unauthorized personnel.

2.3 Discretionary Execution Controls

Discretionary execution controls (DECs) are user defined execution controls. They operate at the user level and give access to files which may be executed. DECs make little or no sense when placed as access controls on non-executable files, but they serve as a compliment to MECs and can be viewed as a logical extension to DAC.

The user can define access controls on most TCBs which indicate read, write and occasionally execute privileges. However, DECs offer an extension, namely the ability to grant, above the standard MEC list, access to their files by other users. The DEC allows the user to add a secondary list of privileged users, which do not conflict with the MEC list. Thus, if user "A" allows user "B" access to his files, user "A" can specify exactly what the user can do. By specifying a DEC of:

```
prog.ru = (MEC = [delete, copy, modify, debug, link, load],
           DEC = [eugen:w, don:w, S5B.g:r])
```

user "A" has effectively said that eugen and don can use all of the MEC features that prog.ru can be accessed with if and only if they contain that feature in their duty specification. S5B.g, however, can only apply those features non-destructively, thus someone in S5B.g applying the delete process against prog.ru would be met with an error, eugen or don would be able to delete it without problem.

You might wonder how this "feature" will help integrity. Using the computer virus [Cohen] as an example, let's examine what would happen if user "A" executed a program from user "B"s directory:

a) User "A" executes program "X" from user "B"

b) "X" contains a virus which notices "prog.ru"

c) it attempts to open the file with update

d) the TCB examines the MEC list to see if "X" is listed as one of the processes privileged enough to access program "prog.ru"

e) the TCB notices that it is a MEC violation, even though user "A" has complete access to his own files!

f) an exception is raised and the program terminated

Why is it a MEC violation? Simply put, access to the program "prog.ru" is restricted to a certain set of known processes. No other processes may be added. Executables may only be modified by trusted processes which implies that ONLY trusted executables could be the carriers of viruses. However, by their very nature -- being trusted -- implies that they can not be carriers and therefore the system is safe. Even if a viral ridden program is installed into the system, the system will remain at a "steady state". Why? Quite simply, in any given operating system only a handful of programs require access to an executable, these being:

load:	the program which initiates execution of an executable on behalf of a known user
link:	the program which establishes the standard MEC for all executables. This is one of the most trusted programs on the system [Wong & Ding].
debugger:	this is optional depending whether or not the debug schema is to be included or not
delete, copy, modify:	these are standard system functions to delete the file from the file system, copy it to another place and to modify those aspects of the file which the TCB allows.

Of course, this is not expected to be a complete or even a correct list, however it is meant to show more clearly exactly what mechanism is being used to protect the system.

2.4 DAC Extension -- The EXECUTE Attribute

Many TCBs do not offer a method of excluding execution of a program for certain individuals. However, the ability to specify what can be read and what written is not sufficient. The simplistic premise that read and write is all that a TCB requires to be safe has been shown to be fatally flawed [Cohen]. However, the ability to indicate that a file is executable but NOT readable or writeable is quite useful. A poorly known operating system CP-6 [CP-6] uses such a method of protection for its system files. Those files that are executable but should not be readable are labeled as such. Thus, whenever a user wishes to list his directory he simply invokes the "l" command but is not capable of viewing the binary. Any such attempt prompts the system to issue an error message indicating the file does not exist. The reason it doesn't exist is that any attempts to read the file don't make sense if the file is executable only. Thus, nothing can read OR write the file.

Using the virus example once more, it is possible to notice that if the virus is halted from being able to enter a file due to the fact that the file is execute only, it only stands to reason that the virus will not be able to propagate. However, if the only restriction was a modifiable DAC, a virus could easily modify the DAC to what it required and then modify it back to its original state.

The inclusion of an execute attribute on the DAC coupled with the MEC would allow the TCB to halt the spread of a virus created by a user of the system.

2.5 Inheritance

In the described system, an executed object becomes a subject and inherits the originating subjects duties and user id. Similarly, for processes run by other users other than the owner but with sufficient privilege, the executing user inherits the MEC list of the accessed process and this supersedes, temporarily, the user's original duties.

3. Dominance Requirements

The dominance requirements for this model is quite different than for MAC or DAC. For DEC, the user invoking the process must be contained within the DEC of the given process:

$$U_{dec} \leq P_{dec}$$

where U_{dec} is the user id

P_{dec} is the process's DEC list.

Similarly for MEC,

duties	U_{duties}	in	P_{mec}	where U_{duties}	is the duties this user can do.
				P_{mec}	is the process's MEC list.

The duties of the user indicate what the user can do with a given process. Anything attempted which does not exist within the user's duty list causes the system to reject the attempted access.

4. Separation of Duty

The method of attaching duties to each and every user and defining them in such a way as to describe their duties fully to the TCB ensures a complete separation of duty.

5. User Enhanced Authentication

The entire PEC process is designed as an enhanced authentication device while at the same time ensuring that illogical data accesses -- writing to a file with an improper subject -- are impossible.

6. Comparison with Clark and Wilson Integrity Mapping

In an as of yet unpublished paper given to me by David Wilson, Clark and Wilson outline a mapping which they see as indicating the level of support offered by an integrity model. This mapping has the following headings. Those features which are fully met with PECs are so indicated:

Prevention of Change	YES
Data Labels and Logs Change Log Change Log with attribution IVP execution for data Domain logs for data	Unknown
Access Control Triple	YES
Application Program Change Control	YES
Uncircumventable User Authentication	YES
Controls on Privileged Users	YES
Dynamic Tracking of Separation of Duties	Possible

7. Conclusion

As one can see, although the system is simple, it could enhance existing systems in the area of integrity to the point that various forms of malicious code could be stopped. Also, unauthorized, Trojan modifications would be easily halted as well since modifications to any given file would be halted by a PEC. No user can modify his own files without operating under the control of a process. This inclusion of an extra layer in which processes operate on your behalf allows the TCB to ensure the integrity of the data and audit those changes when they do happen. Also, the amount of the TCB which must be ensured to be trusted above and beyond the current norm is minimized to the loader and linker. Although processes such as the debugger are dangerous and can modify live files, modifications could be included which ensure that debuggers can not write to the given file. The MEC list could be extended to include read, and write lists as does the DEC.

However, what I hope to have shown is a model of a system which is implementable using Process Execution Controls and which implements the Clark-Wilson model.

8. Bibliography

[Cohen]	Cohen, Fred	Computer Viruses: Theory and Experiments Computers and Security, North-Holland, Amsterdam, vol. 6, no. 1 April 1987, pgs 22 - 35
[Thompson]	Thompson, Ken	Reflections on Trusting Trust Communications of the ACM, vol 27, no 8, August 1984, pgs 761 - 763
[CP-6]		Honeywell Bull CP-6 Programmers Reference Manual, D00
[Sandhu]	Sandhu, Ravi	Transaction Control Expressions for Separation of Duties Fourth Aerospace Computer Security Applications Conference, 1988 pgs 282 - 286
[Wong & Ding]	Wong, Raymond M. and Ding, Y. Eugene	Providing Software Integrity Using Type Managers Fourth Aerospace Computer Security Applications Conference, 1988 pgs 287 - 294

[Clark & Wilson]	Clark, D.D. and Wilson, D.R.	A Comparison of Commercial and Military Computer Security Policies IEEE Symposium on Security and Privacy, 184 - 194 (1987)
[Clark & Wilson]	Clark, D.D. and Wilson, D.R.	Comments on the Integrity Model Preliminary Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), Bentley College, MA, October 1987
[Jueneman]	Jueneman, Robert R.	Integrity Controls for Military and Commercial Applications Fourth Aerospace Computer Security Applications Conference, 1988 pgs 298 - 322
[DoD 85]		Department of Defence Trusted Computer Security Evaluation Criteria, DoD 5200.28-STD, 1985

Naming and Abstraction for Large Security Configurations

Robert W. Baldwin
Tandem Computers
19333 Vallco Parkway, MS 3-04
Cupertino, CA 94014
408/725-7233

If an information system is going to help a site achieve high data integrity the site must have high assurance that the system will enforce the desired access policies. One aspect of that assurance, specifically making sure that the computer will do what it is told, has received a large amount of attention. This note discusses an equally important aspect: assuring that the information system has been told the desired policy. A simple extension to current protection mechanisms is proposed that greatly reduces the difficulty of expressing a high level security policy.

With current protection mechanism, the high level access policies must be expressed as a large collection of low-level statements. Collectively these statements make up the security configuration of the system. As the security configuration gets larger, it becomes harder to maintain and harder to have any assurance that the desired policies are being enforced. For example in a large database with dozens of applications, hundreds of tables and thousands of users, it is hard to manage all the access control lists on the tables. Consider what happens when one of the users gets a promotion. That user may gain some additional access as well as lose some of the access that was available to him. To correctly update the security configuration for such a system, the administrator would need a separate database describing which application programs can be run by users holding a particular job, and which tables are accessed by each application. Basically, security configurations are hard to maintain because of the large gap between the abstractions that form the site specific access policy and the low-level statements understood by the system's protection mechanisms.

This note discusses a simple extension to the ANSI SQL protection mechanisms that allows it to express higher level abstractions. Basically, the extension allows an administrator to group and name collections of privileges to form named protection domains (NPDs). A privilege is either a table privilege like the ability to Insert into the Employee table (i.e., a verb-object pair), or recursively, a privilege can be an NPD. In a conventional RDBMS table privileges are directly granted to users. With this extension table privileges can be granted to a user or to an NPD. Similarly, an NPD can be granted to a user or to another NPD.

This extension allows an administrator to create a directed acyclic graph of privileges that mimic the abstractions found in the high level access policy. For example, a single application program might present a screen that allows a supervisor to create, lookup, and update purchase orders. Clerks use the same application, but accounting clerks can only create orders and shipping clerks can only lookup orders. This policy could be expressed using three domains, two for the two types of clerks and a third domain that can both update orders and exercise all the privileges of the other two domains. That is, the two

clerk domains are granted to (are contained in) the supervisor domain. By giving the domains meaningful names (e.g., order-create, order-super, etc.) it would be easy to grant NPDs to users based on their job descriptions.

The key idea is that naming and grouping reduces the complexity of the security configuration by allowing an administrator to create abstractions. This makes the configuration easier to maintain, and more importantly, it becomes easier to decide whether the high level access policy has been correctly expressed.

To bridge the abstraction gap between the protection mechanism and the site access policy, one must group privileges, not individuals. Groups of privileges correspond to abstract operations that are meaningful in the access policy. Policies may talk about different kinds of users, but these distinctions would be better expressed as attributes of users or their jobs rather than a grouping of users. Notice that the NPD mechanism can be used to group individuals by creating an NPD and granting it to all the individuals in the group. Any privilege can be granted to that group of individuals by granting it to the NPD.

NPDs are similar to both Roles (in the ACF2 sense) and Access Classes (in the BLP sense). A Role is a capability that is given to a user which allows him to use a collection of programs and/or data. Thus, a Role provides a way to name a collection of privileges just like NPDs, but a Role cannot be made up of other Roles so the abstraction tree can only get one level deep. If a role could be granted to another role, then that mechanism would be equivalent to NPDs. Access Classes, like Secret-Crypto, provide a name for a collection of read and write operations that can be performed on a specific set of objects. Access Classes only provide for a single level of abstraction.

Granting a privilege to an NPD and attaching an Access Class label to a document are both operations that must be carefully controlled. The meaning of an NPD is the set of table privileges it directly or indirectly grants, so granting or revoking a privilege from an NPD changes the meaning of that NPD.

The overhead associated with NPDs is similar to the overhead found in any protection system that allows a user to be a member of multiple groups. The set of all possibly accessible NPDs can be computed when a user logs in, and a list of flags can express which ones are currently active (directly or indirectly). The objects being protected would have an access control list that maps user and NPD names into allowed access modes. The total set of allowed access modes between a given subject and object could be computed as the union of all the access modes allowed by any one of the NPDs (note: to enforce certain confinement policies, it is necessary to be able to ignore privileges directly granted to the user, and only accept privileges that have been granted to one of the active NPDs).

Named protection domains are an efficient and simple extension to conventional protection mechanisms. They make large security configurations easier to maintain and verify, and thus greatly improve an information system's ability to support data integrity policies.

The Clark-Wilson Integrity Policy Model as a Model for Trusted Applications

Deborah J. Bodeau
Trusted Computer Systems
The MITRE Corporation
Bedford, MA 01730

The Clark-Wilson integrity policy model describes a policy of (i) maintaining the consistency and correctness of data and (ii) enforcing separation of duties. The Clark-Wilson model represents the behavior of a system in the broad sense, including verification of data stored in a computer system against the real world and certification of processing, as well as the behavior of the computer system itself. While the policy described in the model applies to military systems, the model was developed for commercial applications.

Many commercial operating systems have been or are currently being evaluated against the Trusted Computer System Evaluation Criteria (the "Orange Book"). Orange Book requirements are stated in terms of two complementary descriptions of system behavior: (i) access of named users to named objects, and (for B1 and above) (ii) access of (labeled) subjects to (labeled) objects. Because evaluated systems provide both useful security functionality and the assurance of careful scrutiny, their use in commercial applications is desirable. However, it must be demonstrated that such systems can support a policy appropriate to commercial applications.

At an abstract level, that demonstration involves establishing a clear relation between the Clark-Wilson model and an access control model. One approach is to treat them as unrelated, and to expect each policy to be enforced by the operating system. Since this calls for new operating system mechanisms, it does not allow advantage to be taken of evaluated operating systems. Other approaches have included attempts to show that the Clark-Wilson policy can be enforced within an access control model such as Bell-LaPadula, or within an access control model that uses an alternate label mechanism (Biba). These approaches tend to obscure the key concepts of transaction processing expressed in the Clark-Wilson model; while those concepts may not be visible at the level of operating system mechanisms, they are useful.

An alternative to these approaches is to treat the Clark-Wilson policy as an application policy, to be enforced in addition to an access control policy. An access control policy model would be used to describe the behavior of the system as a whole. A subsystem or application would be trusted to enforce the Clark-Wilson policy, which would apply to defined subsets of the subjects and objects

in the system. Transactions, transaction processes (TPs), and constrained data items (CDIs), as model constructs, would be built from the constructs of the access control policy model where possible.

Work has begun on expressing the Clark-Wilson policy in an application policy model, built on top of an access control policy model. If this approach proves viable, it will provide an example of an application policy model in which the policy to be enforced by a trusted subsystem is different from that of the underlying operating system. It will serve as a basis for exploring how access control mechanisms can be used to support the Clark-Wilson policy, and for determining what types of additional mechanisms are needed.

On the Adequacy of the Clark-Wilson Definition of Integrity

David A. Bonyun

November 1988

AIT Corporation
9 Auriga Drive
Nepean, Ontario
Canada K2E 7T89

I INTRODUCTION

This paper has been written as the result of two completely independent events:

(a) the movement towards the establishment of "security frameworks" by an international standards group (ISO/IEC/JTC-1/SC21/WG1 ad hoc Security in OSI); and

(b) the recent virus attack by a student.

What follows is a statement of concern that NIST work on integrity, ISO work on integrity and the current concern over the obvious vulnerability of many systems to (presumably unintentional) virus activity all be brought into a consistent view of the world.

There are, at the present time, two parallel approaches to the question of integrity. Roughly speaking, these are:

(1) the Clark-Wilson model, first presented at Oakland in 1987 to the IEEE Symposium on Computer Security and Privacy [1]; more recently this work has been and the expansion was reported at Baltimore, October 1988;

(2) the Robert Courtney Strawman generated following 1987 WIPCIS.

Each of these activities has proposed a definition of integrity (or data integrity). Both definitions have tried to focus on the notion of credibility (or expectations) of data; both have decided to bypass the integrity of the system at large. Both definition are probably acceptable to a limited degree, but neither indicates in any immediately recognizable way how it may be used in practice to avoid breaches.

In a larger historical context, a purely methodological approach was proposed in the mid-1970's by Ken Biba [2]: he saw integrity as the dual of security with the corollary that techniques used to ensure security (which means here the relative absence of vulnerability to disclosure) might be modified slightly (or reversed because of duality) to ensure integrity (here implying the relative absence of vulnerability to contamination).

What seems to be at stake is the question as to what one can say about integrity. Is it enough to define the term? Or should something be said about how it can be breached, what measures are applicable to stop certain breaches from occurring, and how to evaluate such measures. The whole subject of integrity of computer systems is expected to be the material in an ISO security framework document and part of the purpose of the present paper is to indicate the kinds of things that are likely to be contained in this framework; the goal is to ensure that NIST and the rest of the security world do not lose sight of the advances in the area that are occurring in other arenas.

II THE WORD "INTEGRITY"

Some years ago I spent a number of weeks chasing the multiple common meanings (and those in more technical and arcane contexts) of the word "risk". To this day there is little unanimity in the definition of "risk" and how the ideas might be related to "threat" but the task of determining its sundry usages was far from futile. Let us try to do the same for "integrity".

The formal (Oxford) definition of "integrity" says, in part:

> "wholeness; soundness; uprightness; honesty' (where "soundness" is "correct; orthodox; logical, well-founded")

Webster's dictionary says, in part:

> "moral soundness; wholeness; completeness, the quality of being unimpaired"

An interesting observation is that "integrity" is an abstract noun used to name an attribute of some entity. The entities to which the attribute can be applied are of some interest and will be discussed below. It is usual to have associated with such attribute names an adjective that can be used directly and more simply; such an adjective does NOT exist in the use of "integrity"; from the definitions above, "sound" might be taken as an approximation to the desired adjective.

It is, perhaps, significant that this workshop deals specifically with "data integrity". Does this mean that the only entity to which the attribute "integrity" ought to be applied while remaining within the terms of reference of the workshop is data? If so, why invite members of the computer security community at large? I submit that in order to provide any measure of assurance that the integrity of data is preserved, the integrity of the system, as a whole, must be considered. The recent virus attack will be used below to substantiate this claim.

Two entities, then, have so far been identified as possibly possessing the attribute of "integrity". These are data (sets) and systems. Clearly the word is used in everyday speech to apply to individuals (implying "honest" or "trustworthy") and in more technical context to a chain or sequence of links (possibly within the realm of communications) where it implies "continuity" or "wholeness".

Of these varied uses of the word, those that deal with data, systems, and communications are the most germane to the present paper. In the security framework document dealing with integrity, a number of different "architectures" must be discussed; the principle contexts are precisely operating systems, database systems, and communications systems.

What is apparent, however, are two points:

(1) the attribute, integrity, means different things in different contexts; and

(2) the different contextual meaning may depend on one another - they are not

independent or orthogonal.

If the meanings are neither identical nor independent, they must be related (not surprising since they all belong to a single noun). But a single contextual definition can be misleading and if it suffers from a lack of operational direction as to how to deal with deviations, then perhaps no definition at all ought to be sought. Before offering an alternative plan of action, I will use the virus incident to indicate the absence of independence among contexts of integrity.

III THE VIRUS

The recent problem involved a program that had the capacity to replicate itself. That the incident in question did not do this replication on top of existing datasets, but rather used free space, makes it possible to argue that "data integrity" was not involved. A comment by someone at Cornell, at the height of the drama, was to the effect that if the program had had as little as two more lines of code it could have destroyed everything in sight. No one, then, would have been able to say that it involved "simply" a breach of availability and not of integrity.

If the system was violated without clear violation of any particular dataset, then we ought to rejoice, rather than ignore the very real possibility of not being so fortunate next time. If the system integrity has been breached, it is almost pure luck (and some intentional restraint on the part of the originator of the virus) that data integrity has not also been breached.

Yet, the definition of data integrity, as proposed, says that data possesses integrity as long as data meets the expectation of those who use it; indicates that the recent incident did not violate data integrity. At no time did the data change so as to fail to meet, in any measurable way, except in terms of availability (is that in the definition?), the expectation of its users. Those concerned with a narrow view of data integrity can breathe more easily now, but they must not ignore the lessons to be learned. Until and unless system integrity can be preserved, then data integrity is at risk.

IV ALTERNATIVE VIEWS OF INTEGRITY

The conclusion I drew above is that a narrow definition of integrity may have negative value. What I propose instead is the operational approach to integrity that asks how local conditions could change so as to threaten whatever notion of integrity is applicable in the context or architecture being considered. The important aspect of this approach is its major underlying assumption: the necessity of preserving the attribute called "integrity" for some specific entities. The nature of the entities will affect the nature of the attribute. But the various ways of removing or reducing the attribute can be studied without having to dwell on these natures; this study can, in turn, lead to appropriate ways of stopping or containing the potential breaches that come to light.

The ISO framework document on integrity will, for each of the three initial contexts (OS, DBMS, Comms) try to deal with what is meant by a breach of integrity. Some examples are:

> OS a modification of one of the internal tables or registers or kernel memory locations by an unauthorized person or by an unexpected method;
>
> DBMS an entry or modification to a dataset that is inconsistent with other data related semantically to it. (DBMS integrity is frequently called "internal consistency");
>
> COMMS a message, broken into pieces by the sender, received at its destination out of order or with missing parts.

I believe that every one of these examples, while constituting a breach of the applicable contextual notion of integrity, falls beyond the definitions proposed by Clark/Wilson and Courtney.

Once possible breaches to integrity have been listed as completely as human beings can anticipate them, techniques for protecting the entities whose integrity attributes are worthy of preserving can be proposed. This is, of course, classical risk assessment and management, applied to integrity. Nevertheless, the waiting for incidents to occur, without anticipatory steps leading to the offsetting of such incidents, is wholly futile even if the time is spent defining what is meant by "integrity".

In their paper at Baltimore, Clark and Wilson provided some details that enhance their original paper; these details do, in my opinion, move in the right direction. These details are concerned primarily with the database context and, one suspects with databases of accounting figures. Nevertheless, they mention:

> (a) prevention of changes (i.e. modifications) to data if the existing material is consistent and faithful to the world it represents;
>
> (b) attribution of change (logging for subsequent auditing) when changes have to be made (see [4]);

(c) constraint of change - the use of well-defined and restricted methods solely as the vehicle of modification; and

(d) partition of change to require more than one actor in the scenario of modification.

While these are all techniques for inhibiting certain kinds of misadventure, they are, it seems to me, carts placed before horses. None of these is capable of holding back a run-away virus. And as all cost something to implement, install and maintain, none is justified by indicating how much of the problem it contains compared with its cost.

Implied by my criticism is the idea that various incidents have different costs associated with them. If I were to imagine the most malicious form of contamination attack, it would be one that was very sporadic and as undetectable as possible - a simple slipped bit from time to time creeping into a datastructure. The detection of the source and the clearing of all contamination would be very different and expensive (indeed, more expensive than restoring fully destroyed data, assuming that backups exist). Evaluation of the effects of various hypothesized incidents can be horrifying, but they can also be used to justify the cost of (at least partially) effective safeguards or countermeasures.

Current research [5] in the realm of risk management involves the use of expert systems to assist in gauging the changes to the risk environment that result from changes to the internal environment (those entities whose integrity and other security-relevant attributes are to be preserved) or to the external environment (those entities that are able to impact internal entities in ways that cause a diminution of the desirable attribute). The determination of changes to the risk environment can now be undertaken without, as before, the request for many committee man-hours of labour. Moreover, the variety of attributes that might be attributed to any internal entity does not include one call "integrity". Breaches to integrity of various entities are defined as events in the risk environment and are evaluated immediately as conditions change in either of the internal of external environments.

To us, this approach is practical and operationally effective. It may lack the nicety of a formal definition if "integrity"; or alternatively, it may possess the quality of permitting different definitions to apply as appropriate to different kinds of entities that may be said to possess integrity.

The framework document mentioned above is aimed at a discussion of issues and concepts within the realm of integrity. These certainly include applicable threats (i.e. ways that breaches to integrity can be realized). In all the framework documents a number of items (besides applicable threats) will be found, and one of these is a discussion of how the particular architecture (or context) provides special considerations in the area of framework. In particular, in the integrity framework, the different perspectives of integrity applicable to mainframes and their operating systems, database systems, and communications systems are examined. The examples in the previous sections of this paper are typical.

There is, as has been noted, some dependency noted between the various contextual concerns for integrity. In a similar vein, there are relationships between the various topics covered by the frameworks. The most major of these relationships occurs between confidentiality and integrity. This relationship is NOT one of duality (as indicated by Ken Biba op cit), but instead indicates that if encipherment is used to achieve confidentiality (for example in communications) then the integrity of the message is also guaranteed.

Various techniques may be used to achieve integrity in different contexts. The enumeration of these techniques is part of the framework; details are considered to be too numerous and are off-loaded to specific technique-specific documents.

A significant part of a framework document is the enumeration of those gaps in our knowledge that require further research to fill. Given an expected (and usual) collection of subheadings for sections of the framework, it quickly becomes apparent where either (a) insufficient information is available, or (b) too many details essential so as to make separate specific papers desirable.

Attached as Annex A is the proposed outline (i.e. section headers) for the integrity framework.

VI CONCLUSIONS

This paper has attempted two things:

(1) to indicate the inadequacy of the restricted point of view of data integrity especially when viruses are considered; and

(2) to provide a summary of work in another standards arena (ISO) also dealing with integrity.

The appeals made in the paper are threefold:

(1) widen your perspective to consider integrity as a much more comprehensive area of concern;

(2) use a risk management approach rather than an analytic one so that potential breaches to integrity can be identified and possible safeguards be implemented; and

(3) keep track of other activity in the realm of integrity - the integrity framework is just one of the other efforts that should be kept in mind.

BIBLIOGRAPHY

[1] Biba, Ken; Integrity Considerations for Secure Computer Systems, ESD-TR-76-372, USAF Electric Systems Division, July 1975.

[2] Clark/Wilson (IEEE Oakland 87)

[3] Bonyun, David A.; Towards a Standard All Purpose Activity Log, TP-4117-80-1A, I.P. Sharp Associates Limited, February 1980

[4] Bonyun, David A.; A New Look at Integrity Policy for Database Management Systems, I.P. Sharp Associates Limited, May 1986.

[5] Bonyun, David A., Jones, Graeme; Expert Systems Approach to the Modelling of Risks in Dynamic Environments, AIT Corporation, November 1988.

ANNEX A

Proposed Outline for Framework Document in Integrity

(for ISO/IEC/JTC-1/SC21/WG1/ad hoc Security in Open Systems)

1. Introduction
2. Scope and Field of Application
3. Reference
4. Definition
5. Notation
6. General Model of Integrity
   - 6.1 Introduction and Historical Note
   - 6.2 Potential Threats to Integrity
   - 6.3 Policy Issues
   - 6.4 Alternative Integrity Mechanisms
7. Management of Integrity
8. Architectural Issues
   - 8.1 Integrity in a Single Open System
   - 8.2 Integrity in Communications Systems (especially, but not limited to, OSI)
   - 8.3 Database Integrity

ACHIEVING INTEGRITY IN AUTOMATED SYSTEMS

BY

NANDER BROWN

U.S. SMALL BUSINESS ADMINISTRATION

COMPUTER SECURITY PROGRAM MANAGER

INTERNAL CONTROLS AND INTEGRITY

Management and providers of computer services should be held to a high standard of care, subject to liability in tort. This common law duty prescribes that providers of computer services should exercise reasonable care in the processing and use of information furnished by a computer before relying on such data [1]. This duty becomes particularly important when reports required by federal laws include data acquired from or processed by automated systems. Equally important are data which may have a direct impact on the life or livelihood of an individual.

Various provisions of the Security Act of 1933 (the 1933 Act) and the Securities Exchange Act of 1934 (the 1934 Act) impose liability for making false or misleading statements of a material fact or for failing to state a material fact. These provisions create a duty on the part of reporting companies to file accurate reports and to maintain accurate records. The Foreign Corrupt Practices Act of 1977 (FCPA) codified this duty to maintain accurate records [2].

Internal auditors and accountants place significant reliance on the system of internal control to provide assurance of the accuracy and reliability of data. The objectives of internal control are to provide management with reasonable, but not absolute, assurance that financial and other resources are safeguarded from unauthorized use or disposition; transactions are executed in accordance with authorizations; financial and statistical records and reports are reliable; applicable laws, regulations and policies are adhered to; and resources are efficiently and effectively managed.

Internal controls should ensure that data produced by a computer are accurate and reliable. This implies that restrictions should be placed on those who have access to computer records and on those who have the authority to enter or alter data in the computer. "Audit trails" should also be used to create documentary evidence of transactions and of who made a particular data entry. In the context of software, internal control is all the formal mechanisms used to ensure the correctness of computer processing and the compliance with management authorizations and standards.

Integrity is just one aspect of internal control that should be addressed by the system of internal controls employed by an organization. Integrity is implemented through the judicious application of internal control standards: (1) general controls; (2) application controls; (3) system control objectives; (4) transaction specific controls; and computer security. Application controls prescribe generic control requirements, whereas transaction controls implement application control requirements through specific practices employed by system owners. See figures 1 and 2 for an illustration of the structure of internal controls.

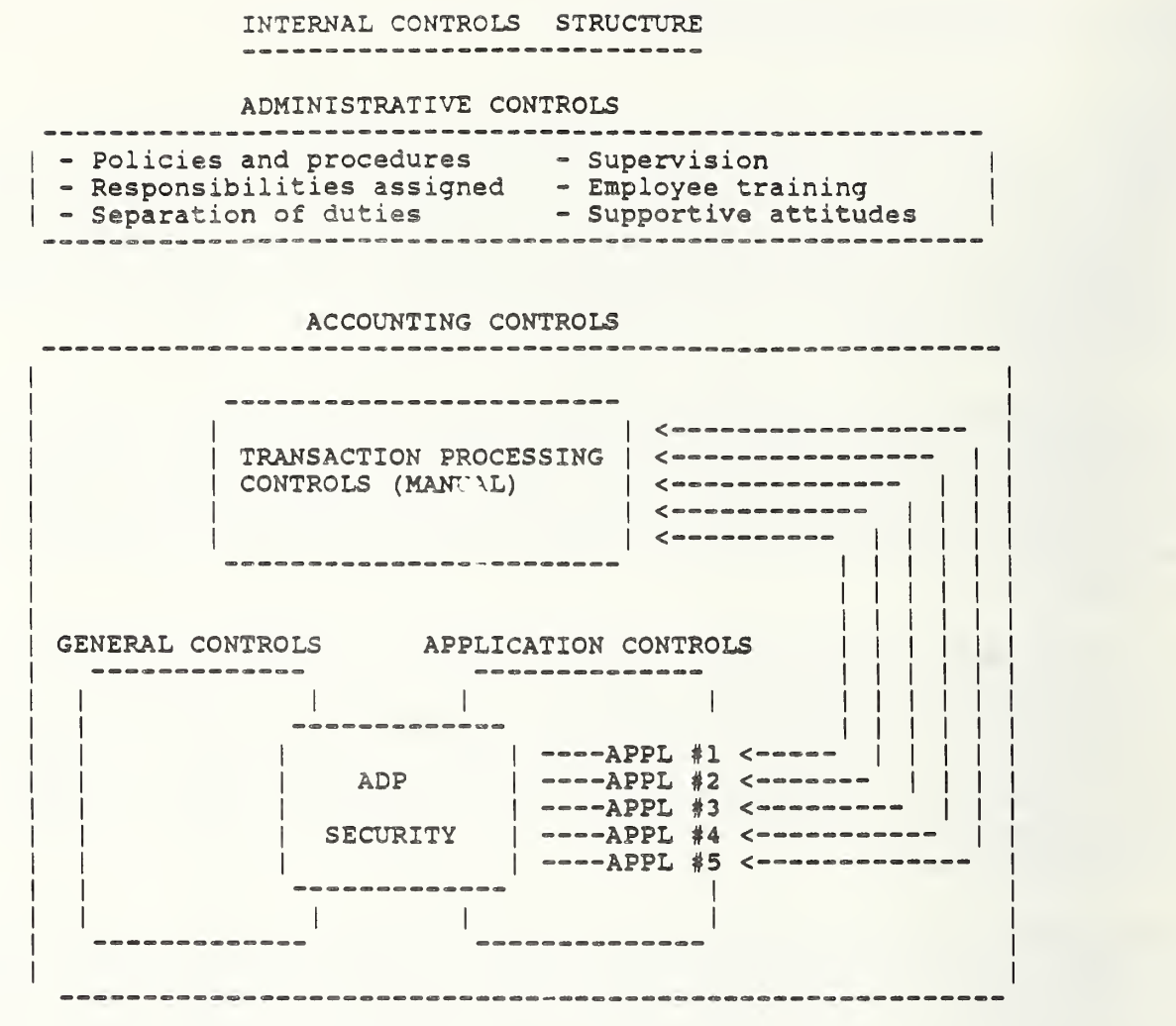
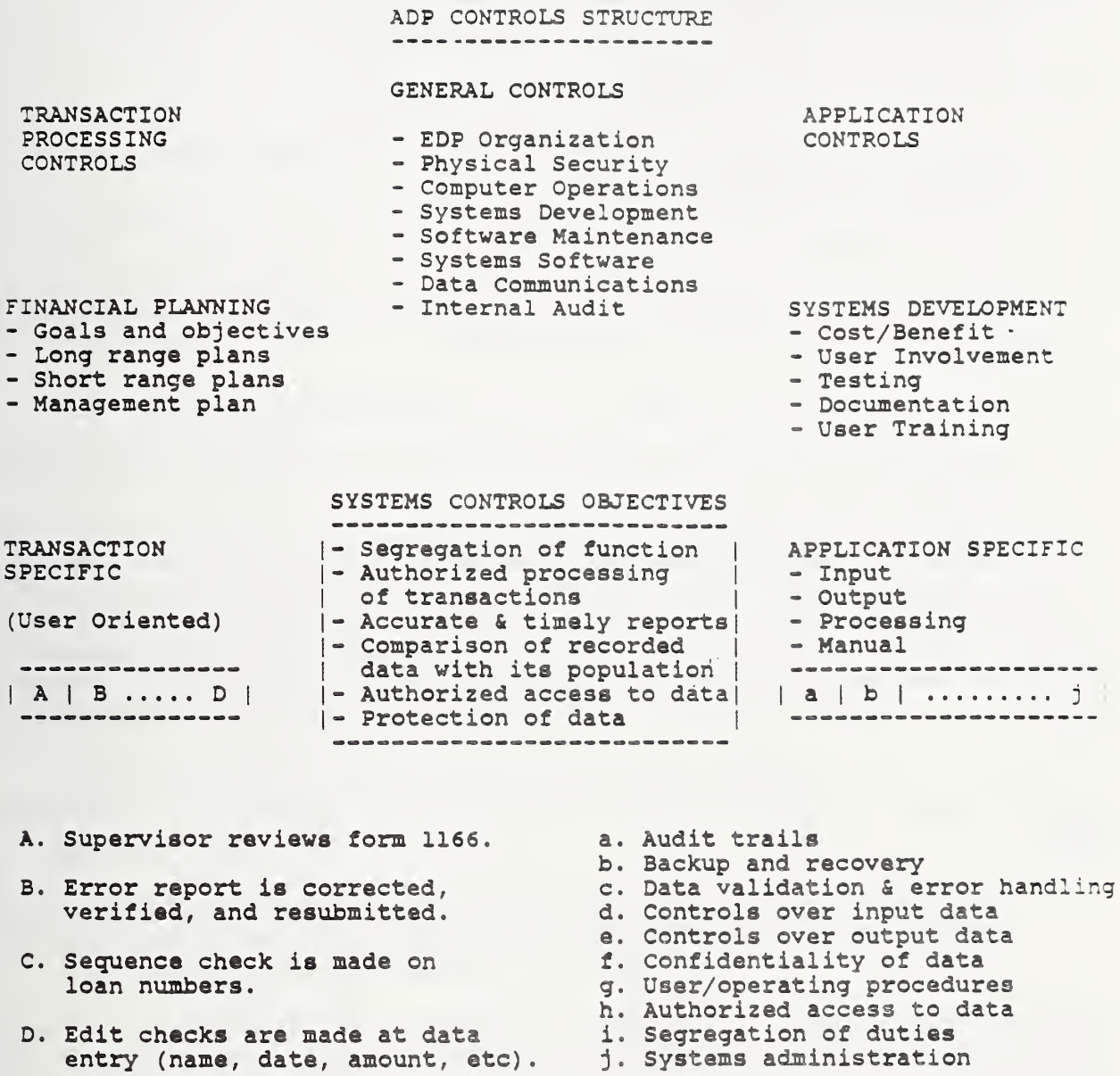


Figure 1. Internal Controls Structure.



A. Supervisor reviews form 1166.

B. Error report is corrected, verified, and resubmitted.

C. Sequence check is made on loan numbers.

D. Edit checks are made at data entry (name, date, amount, etc).

a. Audit trails
b. Backup and recovery
c. Data validation & error handling
d. Controls over input data
e. Controls over output data
f. Confidentiality of data
g. User/operating procedures
h. Authorized access to data
i. Segregation of duties
j. Systems administration

Figure 2. ADP CONTROL STRUCTURE

INTEGRITY DEFINED

Integrity is essentially a descriptive, qualitative assessment of the exactness, purity, and authenticity of a subject. Webster defines integrity as:

> "(1) an unimpaired condition - SOUNDNESS; (2) firm adherence to a code (especially moral or artistic values) - INCORRUPTIBILITY; (3) the quality or the state of being complete or undivided- COMPLETENESS".

In reference to data, integrity can takes on a litany of meanings:

- Data that is processed or maintained on a storage medium is a true representation of the source subject.

- Data is complete and without fault.

- Processed data is accurate, correct, and valid. Automated procedures produce accurate and correct results on valid data items.

- Authorization policies are applied consistently, and reliable results are produced.

- Authorization policies and automated procedures are consistently applied across a broad spectrum for an extended period.

- Processed results are accepted without question.

FIPS Publication 73 defines data integrity as: The state that exists when computerized data is the same as that in the source documents or has been correctly computed from source data and has not been exposed to accidental or malicious alteration or destruction. Erroneous source data and fictitious additions to the data are also considered violations of data integrity [3].

Using the foregoing definitions as input, a definition of system integrity can be stated as follow:

> "Integrity is the appropriate balance and application of computer security, internal controls, and system compliance in order to achieve data and system requirements in support of the goals and objectives of the system"

In the broad sense, integrity may be defined as a requirement for wholeness or completeness. A system can be said to have integrity if it performs according to its specifications.... and does no more or no less. These specifications must state how the system will ensure that its components are functioning properly, and what malfunctions can occur. System integrity is achieved through the combined integrity of its components. In theory, this kind of integrity can be demonstrated logically by inspection or testing. The more complex the system, however, the more difficult it is to achieve system integrity.

INTEGRITY CONCERNS

The problem of integrity is the problem of ensuring that data is accurate. There is a problem in guarding the data against errors caused by the data capturing process and the data update process. Internal control techniques are incorporated to protect the data from errors in data entry, mistakes by the computer operator, and incorrect processing by application software. Security techniques are employed to protect data from deliberate manipulation and inadvertent software errors and system failures.

Data integrity concerns can be summarized in the following categories:

- Unauthorized disclosure,
- Deliberate errors,
- Unintentional errors,
- Errors of omission,
- Misrepresentation of facts, and
- Unauthorized destruction.

System integrity concerns include the broad spectrum of potential problems that can occur in an automated environment. Some concerns are general in nature, and some are related to specific system functions and the mission/purpose of the system. Major concerns include:

- Potential for fraud,
- Potential for law suits,
- Legal violations and non compliance with mandated statutes,
- Inefficient reporting to management,
- Improper authorization, and
- Improper financial decisions.

INFORMATION INTEGRITY AREAS

There are four basic areas in the automated information environment which must be addressed when discussing integrity as a subject of automated systems:

- Data Integrity
- Processing Integrity
- Software Integrity
- System Integrity

Data Integrity

Data has integrity if it exists within defined limits of reliability, and is accurate, consistent, authorized, valid, complete, unambiguous, and processed promptly and according to specifications. Data integrity is achieved through data control measures that ensure correct performance of program functions and error prevention. In addition, these controls ensure that all required data is processed only legitimate data is processed, and no data is distorted or lost.

Data is normally controlled in systems through various programmed data validation procedures, which involve the examination of computerized data to determine if it is accurate, complete, unambiguous, and reasonable.

Errors should be detected and corrected, if necessary, as soon as possible to prevent the propagation of invalid data through the data base. Because the error correction process is quite likely to introduce further errors, data validation methods should be applied thoroughly during the correction process.

In a data base environment, the quality of data is a critical factor. Because data might be recorded only once and thereafter serve as the input for subsequent processing, a high level of quality must be maintained for all elements within the DBMS.

The data base has integrity if the existence, quality, and privacy of recorded data is protected. The data base must be protected from hardware, system software, and application failure as well as from data and user errors.

Processing Integrity

Processing integrity is a necessary component of data integrity. It includes automated procedures and specifications that ensure data are processed in accordance with rules that are documented and approved by users and management.

Processing integrity must be maintained throughout the life of the system. Only users and managers should request changes, and they should have final approval and acceptance before the change is implemented. Auditors should periodically verify that rules are being applied as specified.

Software Integrity

Software integrity is more than program correctness; it requires a program to be correct, robust, and trustworthy. A correct program satisfies its mission, requirements, and specifications. A robust program includes mechanisms for maintaining adequate performance levels during unexpected behavior in the environment. A trustworthy program is well documented, not functionally complex, modular, relatively short, integrated into a rigorous structured architecture, and produced as a result of good programming practices and sensible standards.

Software procedures and information critical to security and integrity should be well identified so it can be more easily audited and protected.

A program should explicitly identify all data elements it accesses, whether they provide data for application purposes or are read to enable the program to cross an access path. The program should not make assumptions about the number of instances of a record type or about the presence or absence of record types.

System Integrity

System integrity is the ability of the system to operate according to specifications even in the face of deliberate attempts to make it behave differently [4]. It includes the combined integrity of all its components (See system integrity model in Figure 3) System integrity also includes the properties of a system and management direction that permit effective development and authorized use; and normally requires a design methodology, structured development, and operational procedures. These requirements include the:

- Use of programming standards,
- Use of documentation standards,
- Use of Software quality control concepts and procedures,
- Preparation and execution of an acceptable test plan,
- Achievement of acceptable test results,
- Availability of user level and program level documentation, and
- Availability of required data and system backup procedures.

In order to ensure system integrity, a secure operating system and system software environment is absolutely necessary. This also includes the availability of a secure communications environment. At a minimum, the operating system must:

- Protect data in memory,
- Protect system software resources from unauthorized access, and
- Perform correct physical input and output processes.

IMPLEMENTING SYSTEMS INTEGRITY

Systems are processing networks that interface with different functions of the organization at various times. Some functions of the organization interface with different parts of the system more frequently that other organizational functions. These networks are abstract as veil as physical, and may be quite complex. These complex systems interface and interact with other systems within and outside of an organization. System integrity is achieved through the combined integrity features of its components. Thus each subsystem, each interfaced system, and processing network must have appropriate and effective integrity components.

Top management support and directions are necessary to ensure that systems integrity assurance objectives are implemented and effective throughout the life of the system.

Implementing integrity in information systems requires a balance in the application of computer security, internal controls, and system compliance requirements. Internal control has been discussed. Brief definitions for computer security and compliance requirements are as follows:

> "Computer security is the protection of automated information against accidental or intentional disclosure, modification, or destruction. It also includes the protection of physical facilities and other computing resources. Three general threats areas are computer processing, natural disasters and personnel".
>
> "System compliance requirement includes mandated Federal and state laws and regulations; organizational policies and procedures; and industry standards that must be met. It also includes special and unique control objectives that must be implemented to protect a given system from unique risks".

Some of the components required to implement systems integrity are illustrated in the matrix listed on the next page.

The matrix below defines some specific techniques for implementing compliance, internal control, and security in automated applications. These techniques are cross referenced to appropriate integrity areas.

INTEGRITY IMPLEMENTATION MATRIX

	Compliance, Internal Control, and Security (C,I,S) Techniques		Data Integrity	Processing Integrity	Software Integrity
A.	Physical security	[S]	X	X	X
B.	Separation of duties	[I]	-	X	X
C.	Required system functions	[C]	X	X	X
D.	Governing regulation	[C]	X	X	X
E.	Standards and procedures	[C,I,S]	-	X	X
F.	Control totals	[I]	X	-	-
G.	Audit trails	[I,S]	X	X	X
H.	Data Validation	[I]	X	-	-
I.	Error correction detection	[I,S]	X	-	-
J.	Run-to-run totals	[I]	X	-	-
K.	Access control	[S]	X	-	-
L.	Password administration	[S]	X	-	-
M.	System testing	[C,I,S]	X	X	X
N.	Data backup procedures	[S]	X	-	-
0.	System contingency plans	[S]	-	X	X

Illustrated below are several application risk areas and various potential adverse conditions that may occur. Risk items are cross referenced to specific control techniques from the matrix above.

POTENTIAL RISKS TO SYSTEMS SECURITY AND INTEGRITY

APPLICATION RISK AREAS	CONTROL TECHNIQUE REFERENCE*
I. ERRONEOUS OR FALSIFIED DATA INPUT	
- Unreasonable data	[F,H,I]
- Fraudulent data	[B,G,J]
- Inaccurate input of source data	[H,I]
- No correction of error transaction	[B,I]
2. MISUSE BY AUTHORIZED END USER	
- Improper distribution of reports	[E,G]
- Management override of application controls	[E,G]
- Conversion of information	[H,B]
- Destruction of data	[E,K,D]
3. UNCONTROLLED SYSTEM ACCESS	
- Theft of data or programs	[A,B,N,O]
- Compromise of passwords	[L,E,G,D]
- No immediate denial of access do to termination	[L]
4. INEFFECTIVE SECURITY PRACTICES FOR THE APPLICATION	
- Poorly defined criteria for Access	[C]
- Careless handling of sensitive data	[E]
- Security reports not used	[E,L]
- Training with production data	[B,E]
5. PROCEDURAL ERRORS WITHIN THE COMPUTER FACILITY	
- Data destroyed during disk reorganization	[E]
- Operator modification of data	[E]
- Operator sabotage of computer	[E]
- Wrong version of program used	[E]
- Wrong file processed	[E]
- Operators do not use safety controls	[E]
- Critical tape files not write protected	[E]
- Mislabeled media erase	[E]
- Tape management inventory inaccurate	[E]

* See compliance, Internal Controls techniques A - O on page lo.

POTENTIAL RISKS TO SYSTEMS SECURITY AND INTEGRITY

APPLICATION RISK AREAS	CONTROL TECHNIQUE REFERENCE
6. APPLICATION SOFTWARE PROBLEMS	
- Inadequate program testing	[E,M]
- Inadequate system testing	[E,M]
- User not involved in systems test	[E,M]
- Testing with production data without security	[E,M]
- Inadequate program change controls	[E,G]
- Inadequate application contingency plans	[C,N,O]
7. OPERATING SYSTEM FLAWS	
- Same file processed by two jobs	[E,J]
- System crash exposes valuable information	[B]
- System crash recovery does not require new access and authentication	[I,K,L]
- User able to get into supervisory mode	[K]
8. COMMUNICATION SYSTEM FAILURES	
- Undetected communication errors result in data errors	[F,I]
- Information transmitted to wrong terminal	[I]
- No positive identification of sender and receiver	[L]

CONCLUSION

In the context of systems and software, integrity has many meanings. However, one meaning that is widely accepted that systems integrity implies quality of information, reliability of service, and confidence in a system"s performance. It is also widely accepted that most systems fall short in meeting these requirements.

This paper has discussed integrity from a generic viewpoint. The result is a better understanding of the many facets of integrity as related to automated systems. One aspect of integrity that is important to remember is that it is a merger of 3 disciplines: internal control, computer security, and functional user requirements. Finally, a model which blended these key elements was presented as an approach to achieving systems integrity. The Appropriate balance in the application of these elements is an overriding system design consideration. It is also important to consider the dynamics and volatility of the system's operating environment.

REFERENCES

I. Computer Crime - Computer Security Techniques, U. S. Department Of Justice, Bureau of Justice statistics, p. Il.

2. Ibid.

3. FIPS Pub 73, Guidelines for security of Computer Application, Department of Commerce, National Bureau Of standards, June 1980.

4. Davida, G. I., "Privacy, security and Database". DATA BASE SECURITY AND INTEGRITY, International Business Machine, 1985, Addison-Wesley Publishing Company, NY.

OTHER INFORMAL COMMENTS ABOUT INTEGRITY AND THE INTEGRITY WORKSHOP

Thomas M Chen
Wang Laboratories, Inc.

INTRODUCTION

The Orange Book's emphasis on preventing unauthorized release of information has forced integrity to receive less attention. While Biba [BIBA77], Boebert-Kane [BOEB85], and most recently Clark-Wilson [CLARK87] have published significant papers, the Department of Defense has yet to embrace integrity policy or controls with the same fervor as lattice models for the prevention of unauthorized release.

Clearly, we need to begin to focus more attention on integrity, and Bob Courtney indicates that an initial task in laying a solid foundation for the upcoming Integrity Workshop is to establish a suitable definition for integrity [COURT88]. Following this sage guidance I went home and tried to come up with a workable set of definitions. Since integrity comes in "different flavors" [PORT85], a proliferation of definitions is somewhat unavoidable. After a period of linguistic wheel spinning, frustration drove me to succumb to the basic principle, "Keep it simple, Stupid" (an approach necessary for those of us willing to admit to our limitations).

While Courtney and Ware's proposed "a priori expectation" integrity definition is a useful concept, it is too broad a definition to assist in identifying system integrity controls. Nevertheless, no matter what definition is eventually assigned to integrity, there exists a set of controls that should receive more attention. If achieving "the earliest practical applicability of the work being done" is indeed a goal of the Workshop, we should avoid a prolonged and futile debate over a definition for integrity.

The bottom line is to identify new integrity controls that will make a system more secure. Taking a simple-minded approach to confine the problem to a manageable exercise, I attempted to identify integrity controls which the Orange Book may not already be requiring vendors to build into their systems.

Without getting hung up over integrity definitions, I considered the following:

THE INTEGRITY WISH LIST

Ability to Prevent Unauthorized Modification

Preventing unauthorized modification of data is an obvious choice for the list and should fall under any integrity definition (including "a priori

expectation"). The security community has already developed policy and mechanisms for preventing unauthorized release, and enforcement of Biba's integrity model [BIBA77] addresses unauthorized modification with a well-understood, natural extension of existing mechanisms for protecting data secrecy. The Integrity Workshop's efforts might be better spent elsewhere.

Ability to Know that an Object is Actually from the Alleged Source

Being able to confirm an object's alleged origin also fits the a priori expectation definition and appears to be a universally desirable integrity property. However, encryption, digital signatures, and trusted path techniques are well-understood solutions. The Integrity Workshop should address an issue in more need of its attention.

Ability to Verify that the System is Behaving Correctly

An a priori expectation is for a system to behave as advertised, and represents a desirable integrity property. While it is true that formal verification techniques are not universally understood, the complexities of predicate calculus (Keep it simple) and a paper by DeMillo, Lipton, and Perlis [DEMIL79] suggest that the Integrity Workshop should not concentrate on theorem provers and verification tools.

Separation of Duties/Least Privilege

Clark and Wilson [CLARK87] have highlighted the desirability of "triples" and separation of duties. Developing mechanisms to enforce these principles is certainly in keeping with preserving a priori expectations. While it is unlikely that one specific integrity policy can apply to all situations, particularly at the application-level; the ability to isolate and protect subsystems to encourage separation of duties and the enforcement of the principle of least privilege appears to be a universally desirable principle in need of more attention.

Lee [LEE88] and Shockley [SHOCK88] have described how to enforce the Clark-Wilson policy using Biba categories. Since systems are currently available that support the Biba model, such approaches are both worthwhile and practical. Nevertheless, it is not clear that the Biba integrity model [BIBA77] was ever intended to cover least privilege and separation of duties. Boebert and Kain [BOEB85] claim that a system using hierarchical integrity compartments to achieve separation of duties "either fails to enforce the desired restrictions or requires an exception from the policy at each step". Their "assured pipelines" and type managers appear to provide a cleaner mapping to Clark & Wilson. In addition to separate domains for creating and isolating trusted subsystems and for enforcing separation of duties, Boebert and Kain included an underlying mechanism for passing data between the domains without violating the kernel's integrity policy.

At the first Integrity Workshop, Boebert demonstrated that assured pipelines and type managers provide a straight forward enforcement of separation of duties as well as other security functionality. For example, type managers could also be used to support enforcement of the Bell-LaPadula model [BELL75].

WORKSHOP RECOMMENDATION

The community has already developed mechanisms and implementations to control write access and to determine if an object is from its alleged source. Formal verification is an art in itself, and is best left to those with "unique" skills and motivations. In order to avoid either treading old ground or getting over extended, the Integrity Workshop could focus on least privilege controls and separation of duties.

Such an approach should include a study of appropriate operating system mechanisms. In fact, one could argue that integrity and disclosure protection might distill down to providing appropriate auditing and isolated protection domains (e.g., triples) with a means of controlling information flow between the domains. Such a mechanism would be sufficiently generic to address both disclosure and integrity, whatever their definitions or policies.

Not getting locked into specific policies avoids forcing applications to conform to security policies that don't quite fit the security needs of their real world environments. Concentrating on mechanisms for separation of duties and least privilege controls could avoid a prolonged debate over terminology and enhance the probability of "the earliest practical applicability" of the Integrity Workshop's efforts.

BELL75 D.E. Bell and L.J. LaPadula, "Computer Security Model: Unified Exposition and Multics Interpretation", ESD-TR-75-306, USAF Electronic Systems Div., Bedford, MA, June 1975.

BIBA77 K.J. Biba, "Integrity Considerations for Secure Computer Systems", USAF Electronic Systems Div., Bedford, MA, ESD-TR-76-372, April 1977.

BOEB85 W.E. Boebert and R.Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies", Proc. 8th National Computer Security Conference, October 1985.

CLARK87 D.D. Clark and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies", Proceedings of the 1987 IEEE Symposium on Security and Privacy, April 1987.

COURT88 R.H. Courtney, Jr., "Some Informal Comments About Integrity and the Integrity Workshop", 1988.

DEMIL79 R.A. DeMillo, R.J.Lipton, and A.J. Perlis, "Social Processes and Proofs of Theorems and Programs", Communications of the ACM, May 1979.

LEE88 T.M.P. Lee, "Using Mandatory Integrity to Enforce Commercial Security", Proceedings of the 1988 IEEE Symposium on Security and Privacy, April 1988.

PORT85 S. Porter and T.S. Arnold, "On the Integrity Problem", Proc. 8th National Computer Security Conference, October 1985.

SHOCK88 W.R. Shockley, "Implementing the Clark/Wilson Integrity Policy Using Current Technology", 11th National Computer Security Conference, October 1988.

Security Protection Based on Mission Criticality

Howard L. Johnson

Information Intelliegence Sciences, Inc.

ABSTRACT

This paper reviews developments and reaches several conclusions:

- o Assurance of service can be achieved as part of the design, thereby making availability in the presence of malicious threat an integrity problem
- o There are two approaches to simultaneously deal with both sensitivity and criticality policies: a restrictive combined data flow policy or a strategy that uses isolation techniques (e.g., encryption)
- o Criticality attacks almost always allow a detection and recovery strategy, which can exploit encoding and is less expensive than a resistance strategy (like sensitivity protection)
- o A DoD security objective and accompanying policy should be based on assuring accomplishment of Nationally critical missions dealing with loss of integrity and denial of service threats
- o When one considers a "Criticality Orange Book" it is found that a large part of the Orange Book criteria is equally applicable to Criticality
- o Future criteria should be built independent of specific security models to the extent possible
- o Sensitivity and criticality issues are present in both DoD commercial security.

Mr. Johnson's position paper, "Security Protection Based on Mission Criticality", appears in the Fourth Aerospace Computer Security Applications Conference. The paper is copywritten by the Institute of Electrical and Electronics Engineers.

Stewart Kowalski

The purpose of this letter is to outline briefly the work we are doing on Data Integrity at Stockholm University.

Our research project is called "Project for System Integrity and Information Security" and is funded in part by the Swedish Department of Defense. The project came into existence on the first of October this year. There are five individuals involved in the project on a part time basis. Two individuals, including myself, are representatives from industry. One individual is from the Swedish Data Inspection Board and the remaining two individuals are University staff members.

Our prime mandate is to examine different system security models being used in North America and Europe and to synthesize and to develop models that would be appropriate for the Swedish society.

In February of 1988 we will be holding a conference to present our preliminary findings and recommendations. It would be beneficial to the project if one of the members was allowed to attend the workshop.

As you are well aware, the modelling of secure systems is not an easy task and much of the first two months of the work in the project has been involved with basic discussion of system security. The remainder of this letter is a quick synopsis of the results of these discussions.

What is in a Name?
System Integrity and Information Security

I believe that the selection of the title for the project can give some insight into the goals of the project.

System Integrity

The project focuses on System Integrity and not "data integrity". In our preliminary discussions on the issue it was clear to us that "integrity" is a system or meta concept. Data can only be correct or incorrect. By using the term "integrity" with data, one can run the risk of missing the forest for the trees.

Information Security

Our project found it necessary that a strong distinction be made between data and information. As to what this distinction is we are still heavily debating the issue but one of the dominant notions is that information is a verb and data is a noun. To quote on of your famous philosophers Buckmaster Fuller "I seems to be a verb".

The commonly used expression here is that information is data which changes you. Thus there is always a strong coupling between information and action. If there is no action there is no information.

These ideas are closely coupled to the Norbets Weiner concept of entropy and thus information is negative entropy. Information security can only be maintained in a system if there is some input into the system that stops the system's natural tendency to entropy.

Of the system models we are examining the one that seems to deal the best with the problem of system entropy (i.e. information security) is the viable system model.

Two of the project participants are working with the viable system model designed by Stafford Beer. Techniques are being developed to perform System Viability Audits. The viability system audit basic claim is that in order for a system to be viable (secure) it must contain five basic subsystems

1) subsystem one is the set of operational elements
2) subsystem two is dedicated to preventing the different operational elements from affecting each other
3) subsystem three maintains internal homeostasis, ensures that the operational units are in fact producing what they are supposed to be producing. (This is a subsystem which is similar to the Clark Wilson IVP.)
4) subsystem four is dedicated to provide a focus of the system's self knowledge.
5) subsystem five's main function is the managing of the tensions between subsystems three and four.

I realize that the above is not as self-explanatory as I would like and it is difficult to image how this model can be used to solve the problem of system integrity but preliminary work in applying this model to a PC security system has shown some promise.

The Projects Practical Approach

We are afraid that the approach outline above is a bit too avant garde for immediate application and in order to deal with Sweden's immediate security problem the existing security models such as Bell LaPadula and Clark Wilson should be synthesized and presented in what we refer to as a "user-friendly-handbook".

The goal of this handbook would not be to specify system integrity as an abstract model but rather to give the end user the tools needed to prove/demonstrate to themselves and to others that they have system integrity. We see a common risk/control analysis approach as an important tool.

The above was a very brief snapshot of the past two months discussions. After the conference in February we shall be able to provide a clearer notion of system integrity and information security.

Stewart Kowalski
Researcher
University of Stockholm
Project for System Integrity and Information Security

Karl Krueger

c/-World Bank Group
Room H2-062
Washington DC 20433

INVITATIONAL WORKSHOP ON DATA INTEGRITY

Position Statement on Data Integrity Issues

From my perspective as the Chief Information Security Officer in a decentralized IS environment of a medium-size, knowledge-based enterprise, I perceive the following issues as important in my information environment:

Continued Integrity of Validated Data over Time

Data can be relatively easily validated at input time against reference information then in force. How do we maintain the validity of the data while the reference information is changing? The data integrity relative to changing validation reference information may be critically affected, or may not be affected at all, or some intermediate level of concern may apply.

Integrity after Restart

In a distributed environment, the same common reference data is used by several processing nodes at the same time. How do you update the information "in phase"? How do you restart the environment after the failure of one or several nodes, so that what has since been updated in another node is first loaded up before resuming processing?

User Awareness of Status of Integrity

How do you share system knowledge about the timeliness, validation and other data qualities with its users, keeping in mind the need to prioritize what you feed them to avoid information overload? This issue refers to both fairly static information at system design level, and to dynamically changing operational information.

Data Integrity Groundrules for all Information Workers

Should the professionals concerned with data integrity prepare a statement of "Data Integrity Groundrules" that can be easily explained, learned and followed by professional and amateur system developers and users. Spreadsheets and their potential for user errors have brought to the attention of auditors the need to follow basic rules to ensure a minimum of integrity in any systems development that needs more than a quarto size sheet to print out!

I am sure we will discuss issues like the Security dimension of data integrity and other issues of the Data Integrity field.

KHKrueger:
M11.IFIP.NISTPos
Original dated 1/18/89
Minor revisions 3/15/89

ISSUES IN DATA INTEGRITY

Jan Kruys, NCR

1. The scope of Data Integrity

   Is this limited to information held within a system or does it also cover interactions (transactions between systems? Does it include both intra- and inter- organizational interactions?

2. The definition of integrity domains, the exercise and delegation of authority within such domains, interchange between domains under different authorities.

   Confidentiality controls are relatively straightforward: they are intended to keep information secret and cooperation between different authorities to effect a common high level policy is feasible.

   With integrity things may be more complicated since the interests of domain owners/authorities need not be identical or even complementary.

3. A clarification of the relationship between data integrity [rules] and access control data flow control.

   Conventional and widely held interpretations of access control focus on confidentiality, not integrity. The simple view that integrity is just the complement of confidentiality has been shown to be untrue and therefore there is reason to assume that access control needs a new interpretation in terms of integrity. The Clark/Wilson approach is one step in that direction.

4. A clarification of the relationship between integrity oriented controls and the security levels spelled out in the 'Criteria'.

   It has been suggested that a Clark/Wilson type policy could be implemented on the basis of e.g. a B level system. This view seems to be overly optimistic in that the B level Criteria imply the enforcement of a mandatory confidentiality policy. This may not leave much room for implementing an integrity policy.

   In addition to this technical relationship there is the political relationship: given the status of the Criteria it may prove difficult to establish a parallel standard for integrity oriented controls. The ownership of such a standard needs to be resolved.

Position Statement

WIPCIS II

Theodore M.P. Lee
Trusted Information Systems, Inc.
Minnetonka, Mn.
9 January 1989

Demurral

This is being written consciously not having re-read the proceedings of the first conference, not having really read Bob Courtney's position paper (although I had heard his impromptu discourse upon it at the last Nat'l Computer Security Conference and did skim the paper enough to decide to forgo reading it until after I had written this), nor having any time in recent memory read any of the fundamental works in the field such as Biba's report. In short, I wanted all my biases to be mine and to reflect what I have truly assimilated about the issues, not just that which was most recently brought to my attention.

On Terminology

Having said that, I do however want to briefly add more confusion to the argument over what the term "integrity" means. (I did see both Bob and Willis Ware expounding at length -- and probably deservedly so -- on the problem of inventing new words etc.; I suspect I will agree with many aspects of their arguments once I have studied them, and probably did when I first printed Willis in the CSF, since I also have from time to time complained about sloppy language in our business, but for the moment I will not really enter the fray.) Bob's definition of integrity is one that is binary: whatever it is, something either has it or it doesn't. That strikes me as being very similar to one of the early notions about "security" (whatever that is) -- early on in this game, one prevailing notion was that something (e.g., a computer system) either was secure or it wasn't; no notion of gradation was admitted. And if anyone recalls, real progress in computer security wasn't made until people accepted that something less than perfection was useful, and hence achievable (if security is binary, since nothing ever is perfectly secure you never move off zero.) In talking about a system's ability to protect against unauthorized modification, and in the confidence we have that people will not attempt to make unauthorized modifications, I similarly believe very strongly that a notion of degree is important, i.e., that the colloquial way of talking about something or someone as have greater

T.M.P. Lee
Trusted Information Systems, Inc.

or lesser integrity is indeed appropriate, even if not linguistically accurate. One of my dictionaries (Britannica edition of Funk & Wagnalls, 1960) even supports that to some extent:

> Integrity n. 1. Uprightness of character; probity; honesty. 2. Unimpaired state; soundness. 3. Undivided or unbroken state; completeness. See synonyms under FIDELITY, JUSTICE, VIRTUE, WORTH.
>
> Fidelity ... 3. Strict adherence to truth or fact; reliability; veracity. 4. Electronics. A measure of the accuracy and freedom from distortion with which a sound reproducing system, such as a radio, will receive and transmit the input signals; it may be high, medium, or low....

Enough of that, suffice it to say that a definition which only says something like "conforms to expectations" (as I recall Bob's) sounds to me reminiscent of the 60's credo that "it doesn't matter what you believe as long as you're sincere" and about as useful in guiding our actions.

Where's the problem?

Again, I want to draw a parallel with the early work on computer security. Although the topic of computer security was discussed a fair amount (much in the vein it still is in the popular press) no great technical or conceptual strides were made until people were shown there was a "show stopper:" -- various proposed ways of using computers could not be allowed because it was patently obvious that to do so, with the then current state of computer security technology, would pose an unacceptable risk that sensitive (classified) information would be made accessible to unauthorized people (enemies.) Furthermore, it was demonstrated that casual, ad hoc, even if well-meaning, technically-competent, attempts to "patch" the holes simply could not be trusted to be sufficient. The awareness that there was a problem of that degree and kind happened only in the defense/intelligence world. I believe strongly that most (but not all) of the commercial world is still in the state the whole world was then: it is not clear to me that very many important commercial information processing activities and networks remain unrealized because implementing them would pose an unacceptable security risk.

So, I must then ask: what commercial, or even government/ DoD/ Intelligence information processing activities remain unimplemented, or are implemented very inefficiently, because the current state of the art in computer integrity is insufficient to adequately reduce the risk that information will be improperly modified? In other words, is the national economy or security sufficiently at risk because we do not have adequate technology to prevent unauthorized (either by accident, sabotage, or subversion) modification of information? Until we can identify what things we feel we cannot do because of some lack in the technology (either laboratory or marketplace) I don't see how we can meaningfully decide what aspects of that

T.M.P. Lee
Trusted Information Systems, Inc.

technology to pursue.

(I do not mean this to start an argument about pure vs. applied research, research vs. development, etc. -- all I'm talking about here is whether their really is an obvious immediate practical problem in need of some kind of fundamental change in knowledge, technology, or social/ business structure -- pursuit of knowledge of even only distantly related truths for their own sake is always good and not what I'm talking about.)

My candidate(s)

First, the problem of protecting against accidental modification of information is a non-problem: solutions to that are well-understood, even if not always used when they should be, and I suspect those solutions cannot be improved upon drastically. (at least, the need to improve upon them is not such as to be the sole justification to search for new or better technologies to do so.) I think that statement even applies when the source of error is human, although perhaps there work on application-specific knowledge-engineering to perform reasonableness checks might be productive.

Second, even though the Clark-Wilson model is interesting (as is the Chinese Wall model to be presented, if two others agreed with my refereeing, in Oakland) I am not convinced that, say, the non-availability of a B2-level implementation of it is anything that is affecting the future well-being of any company or government agency. (One of my long-standing biases comes out here: any security work below B2 fails my definition of a show-stopper-remover.)

So, what's left? Apart from some postulated defense applications that I don't know very much about, which I grant may need better integrity (defense against unauthorized modification of information) than is currently available, I think the virus/trojan horse threat has now reached the point where we collectively have to recognize that the current set of ad hoc measures (and are they ad hoc!) is inadequate. The problem is not that any single corporate data base is so valuable that an attack on it alone carries enough risk and danger to justify heroic measures on a national level but that with the wide-spread use of computers, actually or virtually networked (by exchanging media), there is a low but significant risk (significant because of the consequences) that a well-mounted attack could actually cripple major portions of the national economy all at once. I believe strongly we know what the technology is to solve the problem (configuration control through integrity labels) but the societal, economic, legal, and regulatory aspects of putting that in place are formidable. (I disagree with a comment Bill Murray made on the Virus Forum: self-regulation won't work -- the equivalent of a government certification in this arena is just as necessary as in others covered now by FDA, NBS, FAA, UL, etc. -- {treating UL as a quasi-governmental activity.}) It is also true that even though we know a technical solution, we don't know the operational consequences of employing it.

T.M.P. Lee
Trusted Information Systems, Inc.

Miscellaneous Thoughts on Mandatory Integrity Policies

Amplifying on the last few sentences above, I believe it important to see if the mandatory integrity model can indeed be fleshed out to a usable solution. (There are loose ends in all the variations on the theme.) Several systems have implemented it, but it would be misleading to say that anyone has any real experience with it and certainly there has been little progress at all in applying the principles in a networking/ data exchange environment, which is where it is really needed. (The SAT/LOCK type enforcement mechanism may be equivalent, but I have reservations about its seeming rigidity and apparent lack of any notion of import/export of integrity labels.)

Although the mathematical duality between the mandatory security and mandatory integrity models is attractive in itself, examining that duality further in fact leads to, for me anyway, interesting insights. One of the first observations one makes about the integrity model is that as soon as you implement it in a system, almost all the code in the system (e.g., terminal handlers, editors, mail systems) has to have suitable integrity levels and/or categories attached to them or nothing will work, i.e., in some sense the boundary of an "integrity TCB" includes everything and one finds almost immediately the need for a lot of trusted (with respect to integrity) subjects. That shouldn't be surprising and the only reason we don't notice a similar phenomenon in the dual case is that almost none of the code in a system is classified, certainly not the operating system and third-party applications packages. If, on the other hand, the vendor of a system were to really regard his code as proprietary and to give it a mandatory security label enforcing that proprietariness, and, say, the vendor of the mail system and of the data management system were to do likewise, with two more labels, the way in which one would have to set up and run such a system would not be a lot different in character from the way one would have to run a system with the "right" integrity labels.

Another parallel is that associated with both models are "impedance matching" transforms. Cryptography transforms information of high sensitivity to information of low sensitivity. Deliberate redundancy in the form of error-correcting codes and strong-checksumming transforms information of high frangibility (i.e., information needing careful handling) into information of low frangibility (able to be folded, stapled, and mutilated.) (One could probably even pursue dualities between the transforms: spread-spectrum transmission for security seems at least an in informal dual of retransmissions protocols for integrity.) It is interesting, and perhaps there even is a reason for it, that in both cases the art of using such transforms is quite advanced -- only recently has the ability to protect information without the use of the transforms been developed for security, and, as noted above, one can't quite yet say that has been done at all with respect to integrity in any practical way.

T.M.P. Lee
Trusted Information Systems, Inc.

A final parallel (or, maybe, antiparallel). For some reason, as soon as people think about information integrity they want to immediately take a fine-grained view: rather than trying to protect an entire file they want to distinguish which parts of it need to be protected to what extent from whom against what kinds of errors, sabotage, or malicious alteration. I don't know, since it was before my time, whether the very very early discussions of information security went down that path, but certainly the earliest public ideas (e.g., the Ware Report, the various Mitre papers) had taken a coarse-grained (entire file, paper, etc.) view; had they instead insisted on the fine-grained view I'm not sure the progress that has been made would have been. I wonder if there isn't a lesson there.

(Yes, I can hear the counter-argument: "the whole field of computer security in sticking to such simplified views of the world has led to a technology of very limited utility; let's start this data integrity business off on the right foot by diving in and tackling the hardest problems we can find -- that way what is done will be applicable to real-world commercial problems.") But, when?

On Terminology (encore une fois)

Guess I can't force myself to stay away from the terminology issue after-all. A couple of observations. It's probably too late to close the barn door. The security (confidentiality) policy side of the dual is already sloppy and confused. We have "security", "sensitivity", "classification", and "clearance" levels and labels, all meaning roughly the same thing. And we also have "compartment" and "category" (and "access approval" as distinguished from "clearance".) At various times there have been doomed attempts to be precise about these things and to have each of the above terms have a specific, different meaning. (I'm not sure that the distinction between "security level" and "sensitivity level" was ever precise in anyone's mind, but at one time there were definitions of the two different enough to be useful to a Criteria Lawyer; "clearance" and "classification" have always been fairly precisely understood by anyone having to deal with classified information to a reasonable extent, even if they get confused by more ordinary folk, although as soon as compartmentation gets introduced even the cognoscenti get muddled.)

A further complication is that the TCSEC has quite explicitly appropriated the term "security" to refer only to confidentiality. (See the mandatory access control policy paragraphs.) For as long as I've been in this business, however, the term "security" always included the three aspects of confidentiality (protection against unauthorized viewing of information), integrity (protection against unauthorized modification of information and availability (protection against unauthorized denial of service.) It would be interesting to find the first appearance in print of that triad.

In any case, when dealing with information integrity, the

confidentiality qua security dual and the model of the TCSEC leads one, in my mind, to recognize there are three aspects of the topic that ought to have separate terms but which can be (or, at least, are, in colloquial usage) informally all lumped under the single term "integrity":

-- a measure of the consequences of unauthorized modification of the information in question (analogue of classification or compartmentation) -- note that this is only indirectly related to the reasons why unauthorized modification is bad, in the same sense that the reasons for classifying information at a certain level can vary greatly.

-- confidence that a particular person can be trusted to enter or modify a particular kind of information (analogue of clearance or access approval.)

-- assurance that a particular program (in a particular system, operated under particular rules, etc.) given particular inputs (including those of its users) can be entrusted to modify that information properly. (analogue of evaluation class)

Note that the principle of compartmentation in the confidentiality world has a useful parallel in the integrity world: a person (or, a program) is only entrusted to deal with a particular kind of information

a) if one "understands" the consequences of dealing with that information, (has been properly programmed to process it) and,

b) it has been determined that one needs to have access to that information (must use it for its computations)

DATA INTEGRITY ISSUES

POSITION STATEMENT

01-03-1989

Kurt H. Meiser

Coopers & Lybrand
Auditing Directorate
1251 Avenue of the Americas
New York, New York 10020
Phone (212) 903-3221

CONTENTS

1.0 Background and Limitations

This position paper summarizes my position on data integrity. It has been written within certain limitations and is based on practical experience in specific system environments, as outlined below:

1.1 Information Security Terminology

While writing this statement, I do not have access to the Strawman paper or any other related document, e.g. the Clark-Wilson paper. My "working terminology" may conflict with terms and definitions agreed and used in other discussions and papers.

1.2 Commercial Computing Environment

I have gained my data security and integrity experience strictly in commercial computing environments, within and outside the U.S.

1.3 System Perspective

These integrity considerations are discussed from an operating system perspective (as opposed to networking or DB/DC) in the context of the following specific software environments:

Operating System

The operating system discussed is MVS, IBM's premier operating system for large commercial main frame systems. I have recently performed a great number of security/integrity reviews in this environment.

Access Control Systems

The following discussion is based on the functionality currently available in the major access control systems for MVS, CA-ACF2, RACF and CA-Top Secret.

2.0 MVS Integrity

Integrity in an MVS system consists, in my view, of three different types of integrity which are achieved or addressed at three different system levels or layers:

- Basic data integrity - the quality of stored data elements
- System integrity - the separation of system and user programs
- Application integrity - the quality of application data and programs

2.1 Basic Data Integrity

Basic data integrity is the system property that guarantees the quality of basic data elements handled by the system, e.g. data fields in main storage, data records written by DASD, etc.

Computer users have, based on this property, the expectation that data presented by the system to their programs is good (identical with the data received by the system during previous operations) unless error conditions are raised.

The responsibility for basic data integrity is with hardware and software (operating system) vendors.

It is generally assumed that today large computing systems have been extensively tested by the vendor and have a high degree of basic data integrity when released.

2.2 System Integrity

MVS system integrity, as announced by IBM, is the assurance that programs that are not specifically authorized cannot assume system privileges or alter/bypass storage protection or access controls.

Based on this property, system users have the expectation that the system will guarantee address space isolation and has the ability to enforce (RACF/ACF2/Top Secret) access control rules.

The responsibility for system integrity is with the computer installation, because the major mechanisms for system integrity are under customer control.

Experience shows that, although the necessary controls are available, currently, most MVS installations do not use them properly to achieve a high degree of system integrity.

2.3 Application Integrity

Application integrity is the quality of data processed and maintained in an application.

Application users have the expectation that data is only processed in a controlled way and that it represents reality.

In my view, no single well defined set of functions and controls to achieve application integrity is currently available. The following sections will discuss requirements, available controls and common exposures in this area.

3.0 Application Integrity Requirements

The three elements necessary to achieve application integrity are, following the thoughts of the Clark/Wilson model, file integrity, program integrity and verification procedures.

3.1 File Integrity

Application file integrity exists when all files of an application have valid states; this requires that, once a file has a valid state (determined by adequate verification procedures), it is only manipulated by validated programs (TPs) which, by definition, leave the file in a valid state.

Controls are required to allow the definition and enforcement of file/program relations.

3.2 Program Integrity

Program integrity exists when an executable program cannot be changed or replaced (invisibly) without changing its external characteristics such as name, version, level, etc.

Controls are necessary to protect executable programs from any hidden changes or, at minimum, to automatically detect modifications and prevent the execution of a modified program.

3.3 Verification Procedures

Verification techniques and procedures are necessary to establish (and re-verify periodically) the validity of data and programs.

Controls are required to identify (tag) the quality of data and to prompt automatically for a re-verification.

4.0 File Integrity Considerations

In an MVS environment with access control software installed, file integrity can be addressed in the following way:

4.1 Available Controls

The major access control packages offer today the necessary mechanisms to grant and limit access to files through specific programs (data pathing, conditional access, etc.).

4.2 Typical Exposures

A variety of access control privileges such as OPERATIONS and NON-CNCL, often granted to a large number of users, bypass the above controls.

4.3 Evaluation

Basic (discretionary) controls exist; they can be bypassed by privileged users and are often difficult to implement and maintain. True system enforcement would require easy-to-use mandatory controls.

5.0 Program Integrity Considerations

In an MVS environment with access control software installed, program integrity is a major concern because programs are, in general, treated as data rather than as separate and specific objects.

5.1 Available Controls

Access controls are implemented at the library level; there are no practical controls available for the protection of executable programs directly. Although rough manual audit techniques exist, they do not qualify, in my opinion, as control mechanisms.

5.2 Typical Exposures

Executable programs, like any other data, can be accessed and modified through standard access methods, and no meaningful audit trail is generated by the system. Superzap is only one convenient IBM-supplied utility program for this purpose which in fact does create a limited audit trail.

5.3 Evaluation

With currently available techniques, program integrity can practically not be achieved in MVS. It does, however, not appear too difficult to implement adequate controls in MVS and the access control packages. Cryptographic authentication techniques might lend themselves to good technical solutions for this problem.

6.0 Verification Considerations

The validity of data and programs must be established initially, and re-verified periodically or after events have occurred that might have invalidated the information.

6.1 Available Controls

I am not aware of any automated controls or mechanisms to manage the validity and quality of information other than through basic format checking. Although manual inventory and control procedures have been developed in many varieties, there is still a need for system enforced controls.

6.2 Typical Exposures

Data quality may be inadequate in the sense of a stricter definition of data integrity than commonly used today, because the information may never have been verified initially, after being processed by unknown programs, or after an appropriate period of time.

6.3 Evaluation

Techniques must be defined and developed for system-managed information verification. More research appears to be necessary before software implementations can be considered. In my view, these techniques must involve external interfaces (e.g. with an auditor) as well as the monitoring of the programs and processes handling the information.

7.0 Observations and Conclusions

In summary, these are my observations and conclusions (from a systems perspective):

7.1 Definitions and Models

Whilst commonly accepted definitions and models are available for the protection of the confidentiality of data, initial data integrity models exist basically in their development and discussion states.

7.2 Products and Features

Similarly, security products in the MVS arena contain good data access controls (from a confidentiality point of view), while offering only few and incomplete features for managing data integrity. MVS provides mechanisms to establish system integrity as a basis for all other software based controls.

7.3 Commercial User Community

It is my experience from a large number of MVS security and integrity reviews in the commercial environment that, in a technical sense, the majority of installations have, so far, not adequately addressed system integrity issues nor properly implemented their access control software.

Executive management, however, is usually unaware of such problems and even assumes that their computers have good application integrity.

DATA INTEGRITY ISSUES

for

1989 INVITATIONAL WORKSHOP ON DATA INTEGRITY

submitted by: Dale W. Miller

Data integrity in a commercial organization is very much dependent upon controlling changes to the application programs and operating systems that process that data. A method of change control based on the Clark and Wilson Model can possibly be used to define the roles of all personnel involved in the process of installing or modifying applications in commercial organizations.

Businesses lose a considerable amount of time and money resolving problems caused by changes in the software that are not expected by the user of the information. These changes in the software, whether deliberate or accidental, usually affect the integrity of the information. Often the impact on the integrity of the organization's data is not fully discovered until costly decisions, processes, or transactions have been completed on the basis of that information.

I believe that in order to achieve DATA INTEGRITY as defined by the INTEGRITY WORKING GROUP, (and I agree with that definition), in a large commercial organization, it is first necessary to achieve data integrity for the application software and operating system software that is used to process the organization's data.

The difficulty of achieving integrity is compounded by the fact that many of the application systems are large scale integrated systems. This means that diverse subsystems often have an impact on the information and it is not easy to establish bounds for applying change control rules. Application development teams in commercial organizations use a variety of application development methodologies, many of which are not rigorous enough to enforce rules which would ensure data integrity.

I believe it will be helpful if the Workshop can address the integrity of software as a prerequisite for the integrity of data. It seems that the Clark and Wilson Model can possibly serve as the basis for the development of a software change control structure which will contribute to achieving integrity in application software and eventually in the data processed by that software.

Integrity Isn't Black or White

Submitted by Lee Ohringer
Chairman
DC/ACM SIGSAC

Trying to define computer system integrity is similar to having the blind men describe the elephant. Each person tends to define the total concept in terms most familiar to the part of the problem with which he/she is most familiar. As a result, we may end up with only a partial definition of the problem that we are addressing.

A computer system is a combination of at least the following: hardware, systems software, applications software, data, the physical environment, and people. It then follows that computer system integrity is some combination of the integrity of the components of the computer system. (But security should also be one of the parts of every computer system. It is therefore quite possible that integrity is not part of security but instead integrity is a consequence of good security and other factors. Further discussion of this possibility is left for another time.)

The purpose of this paper is to discuss data integrity issues. I therefore focus on sections of the Strawman that address data integrity directly. On page 3, the third paragraph from the bottom contrasts data quality and data integrity. I agree with the Strawman that this is an important distinction to make. Integrity is not the same as quality.

The Strawman identifies the following attributes of data: accuracy, timeliness and completeness. I would add consistency, source, reliability, correctness, structure, and precision as relevant properties. Of these, consistency seems to be particularly relevant to a discussion of integrity. It is different from the other items in that (1) it seems more likely to apply to a collection of data rather than to a single entry, and (2) it is the only item on the list that, by itself, can destroy the integrity of the data. i.e. If the data is inconsistent, then regardless of what can be said for its other properties, the data will lack integrity. I therefore propose that consistency is a necessary, although maybe not a sufficient, component of integrity. Identifying other necessary conditions and sufficient conditions for integrity will help us broaden our understanding of integrity and could provide us with means for achieving it.

In spite of the questions that I have raised, I can accept the definition of integrity proposed by the Strawman and do so in order to move on to more important matters. I do emphasize however that the Strawman definition does include the words "meets an a priori expectation of quality that is .... adequate." The inclusion of the concept of quality in the definition, albeit limited, is necessary so that our definition of integrity corresponds to general accepted notion of the term.

The point where I have a real problem with the Strawman is when it states "that integrity has only two states. There are no degrees of integrity."

At this meeting we are addressing data security. I offer the following scenario as a basis for considering degrees of integrity rather than having it be either present or absent. Suppose we begin with a computer system that by the Strawman's definition has integrity. Then suppose that some one-time event occurs that changes some data in an isolated portion of the data base. In most ways the system continues to perform as before, but occasionally gives unexpected, unexplained results. For example, we could have a payroll system in which the zip codes for some payees were altered. There is no denying that the integrity of the data has been changed. But I contend that it would be counterproductive to say that the data, and therefore the entire computer system, no longer has any integrity. I would prefer to say that the data now has less integrity than before, but still has high integrity. For me, this system would still be preferable to a system with moderate, low, or even unknown integrity.

Because of these kinds of considerations, I think that models in which the measure of integrity is based on a continuum will provide us with a more useful concept for measuring security than those models where the concept of integrity is based on a binary value. It is for this purely practical reason that I urge the rejection of the "all or nothing" concept of integrity put forth in the Strawman.

INTEGRITY CONCEPTS WITHIN THE FRAMEWORK OF INFORMATION SECURITY

Position Paper
January 23, 1989

Donn B. Parker
SRI International
Menlo Park, California

Introduction

Several definitions of information integrity are unsatisfactory to me because they do not address the roles of stakeholders (those who use and are affected by the definition), are not set in the context of the definitions of the other purposes of information security, and do not recognize the intrinsic integrity attributes. The level of abstraction of the definition of information integrity also has a bearing on its scope of applicability. The integrity of the systems that process the information and the integrity of the enterprise of organization itself would be treated differently, and the definition would be put in a different context and might be changed. (I will address this subject in an extension of this paper; I believe it is a sufficiently important concept to mention here, however.)

Stakeholders and Their Roles

It is generally agreed that three distinct purposes of information security exist: preserving the integrity, maintaining the confidentiality, and ensuring the availability of the information assets of an organization. Information security is the responsibility of the people who have authority over the information and are held accountable for its prudent care. Information security professionals, technicians, and scientists (*security specialists*) are responsible for advising the users in security, providing the users with controls and practices, and for safeguarding the information in their own direct care, such as audit logs and control specifications. Because security specialists are advisors and providers, I call users, custodians, and owners of information their *clients*.

The sources of information loss are people--with and without intent--and forces; I call these, collectively, the *enemy*. Clients are responsible for protecting their information from the enemy with the advice and assistance of security specialists. Clients may delegate the responsibility for protecting some of their information to security specialists, in which case the specialists have the dual role of client and consultant.

The fourth stakeholder in security is the *auditor*. Auditors report on the state of and deficiencies in security, and they may be internal or external auditors or regulatory agents.

I believe information integrity concepts must be considered with respect to all four of these stakeholders. Clients view integrity as a state to be achieved relative to their and others' information use and associate integrity

with quality. Security specialists attempt to learn the integrity needs of their clients in a clear, precise way, and are not involved in clients' quality concerns. They can then analyze the needs relative to the potential threats from enemies (also based on input from clients.) The results help them offer controls and security practices that will contribute to the preservation of the integrity requirements of the client, be effective against the enemy, and meet with the favor of the auditor. The security specialist does not engage in or advise the change or improvement of the clients' information quality or integrity as a primary concern; that is usually outside the specialist's area of expertise and authority.

Auditors view integrity from the perspective of meeting standards, prudent care, and regulatory or legal requirements where, again, integrity is expressed in terms of commonly accepted intrinsic attributes. The accidental enemy views integrity from the perspective of ways to avoid penalties of accidentally violating integrity attributes. The intentional enemy is concerned with the work factors, risks, and means of avoiding impediments in meeting his goals of violating integrity attributes. Given the client's fixed integrity requirements, the enemy's view of integrity is the most important for the security specialist in determining necessary controls and practices.

In summary, integrity is binary (exists or it does not) and invariant as required by clients, and acknowledged and preserved by security specialists. However, integrity is variable as defended by security specialists with controls and attacked by the enemy in terms of work factor, motive, strategy, and other factors.

Relationship of Integrity to Confidentiality and Availability

Information security concepts must also be considered with respect to the two other purposes of security (maintaining confidentiality and ensuring availability), because integrity controls and practices affect the confidentiality and availability of the information. This relationship is not discussed further here. However, definitions of confidentiality and availability are included in the next section to demonstrate and emphasize the need to differentiate among intrinsic, extrinsic, and external attributes among the three purposes of security.

Definitions

The three purposes of information security are defined below in terms of the stakeholders' use of them (especially as specified by clients and preserved by security specialists).

- Information integrity is the intrinsic completeness, validity, wholeness, correctness, timeliness, accuracy, consistency, and precision of the information.

- Information confidentiality is the extrinsic secrecy of the information (who is allowed to know or possess and not know or possess the information and under what circumstances).

- Information availability is the degree of assurance that information will be where, when, and in the form it is required to be in terms of external conditions.

Intrinsic, Extrinsic, and External Attributes

Consistency with common use and dictionary definitions of integrity, confidentiality, and availability requires that integrity be limited to intrinsic attributes of information, including those identified in the Clark-Wilson paper of internal consistency (e.g., completeness, wholeness) and good correspondence to real-world expectations (e.g., accuracy, correctness, and precision). Attributes of confidentiality are extrinsic. Confidentiality is independent of internal attributes. It concerns such attributes as lists of people authorized to read it, forms of the information that make it unintelligible to some people and readable to others, and labeling its level of sensitivity. Availability attributes are external conditions to ensure the correct form of the information is where it is supposed to be and when. These external attributes are different from the extrinsic attributes of confidentiality. The confidentiality attributes focus on access to the information, whereas availability attributes pertain to places and time.

References to Integrity, Confidentiality, and Availability

- "Data Security--It's Evolution and the Job at Hand," William H. Murray, Computer Security Journal, Vol. V, No. 1, 1988 (Computer Security Institute, Publisher).

  "Data security can be defined as provision for the integrity and confidentiality of data.. . . "

  Robert H. Courtney, Jr., says that "data has integrity when it is as good as we think it is."
  ". . . data can be said to have integrity when it agrees with that which it is supposed to describe."

- "Computers: Crimes, Clues and Controls," Prevention Committee, Presidents' Council on Integrity and Efficiency, March 1986.

  Page 9: "Information security provides assurances that the following are achieved:
  - Confidentiality of sensitive information
  - Integrity of information and the related processes (origination, input, processing, and output).
  - Availability of information when needed; and
  - Accountability of the related information processes."

- Building a Secure Computer System, Morrie Gasser, Van Nostrand Reinhold, 1988.
  "Overview. What is Computer Security? Secrecy, integrity, and denial of service."

- "Security In Open Systems: Security Framework for the Application Layer of Open Systems," European Computer Manufacturers Association, December 1987.

  Page 2--"The task of the Security Facilities is twofold:
  - to protect the integrity and confidentiality of Security Objects (data and programs) in the logical sense, e.g., limiting access to certain users and applications.
  - to protect the integrity and confidentiality of Security Objects (data and programs) where they are subject to physical or other external attacks, e.g., when stored on magnetic media or when transmitted over communications links."

- Trusted Network Interpretation (Red Book, NCSC-TG-005, Version 1), National Computer Security Center, July 31, 1987.

  Other Security Services:
  1) Communications Integrity
     Data origin authentication and field integrity
  2) Denial of service prevention
     Continuity, protocols, and network management
  3) Compromise protection
     Data and traffic flow confidentiality and selective routing

- Information Systems: Agencies Overlook Security Controls During Development, U.S. General Accounting Office (GAO/IMTEC-88-11), May 1988

  Glossary
  Data Integrity: The state that exists when computerized data are the same as that in the source document and have not been exposed to accidental or malicious alteration or destruction.

(2) The property that data have not been exposed to accidental or malicious alteration or destruction.

(3) In a data base system, avoidance of simultaneous update where two concurrently executing transactions, each correct in itself, may interfere with each other so as to produce incorrect results.

FIGURE 1 INFORMATION SECURITY FRAMEWORK

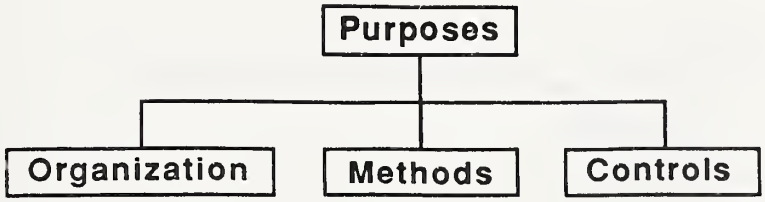


FIGURE 2 VULNERABILITY MODEL

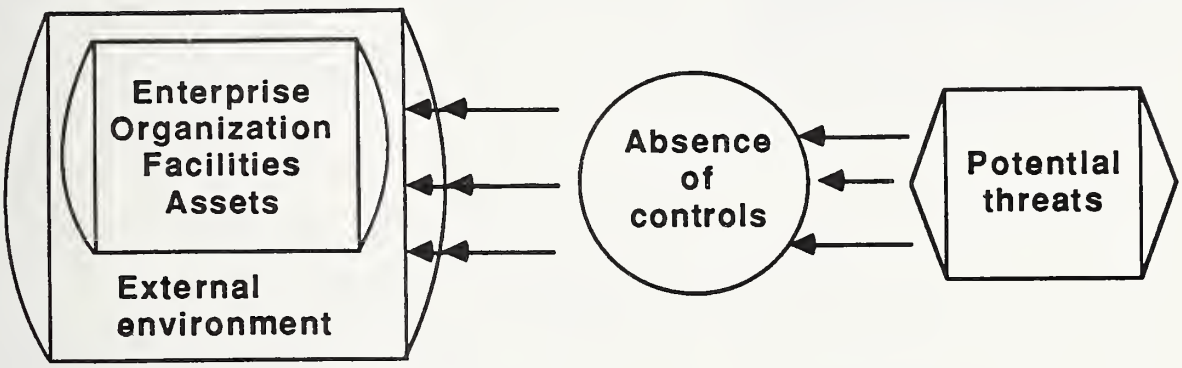


FIGURE 3 FRAMEWORK PURPOSES

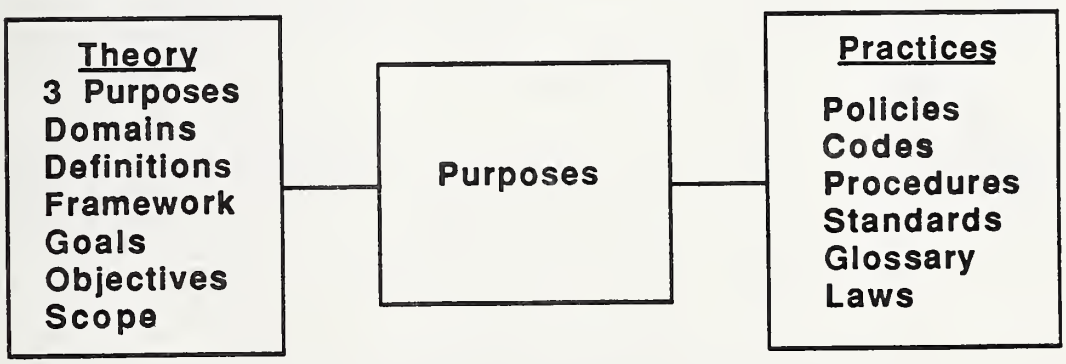


FIGURE 6 FRAMEWORK CONTROLS

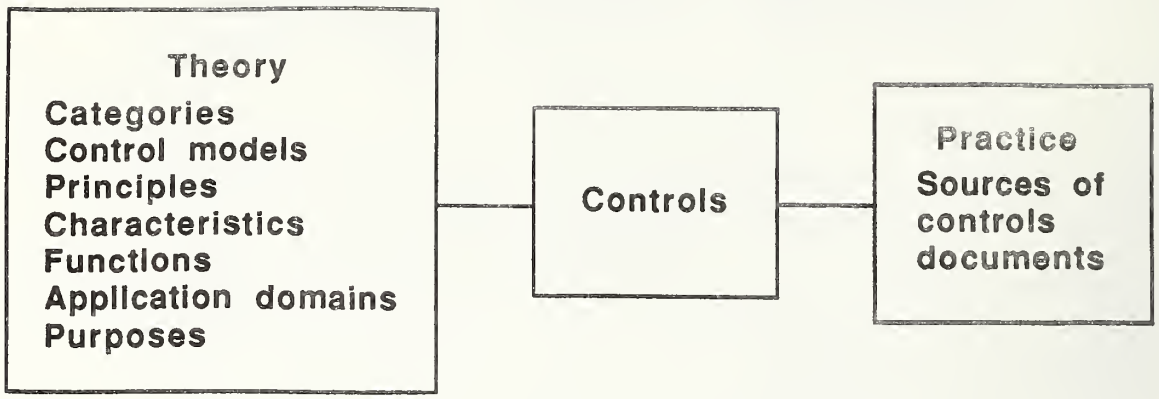


FIGURE 4 FRAMEWORK ORGANIZATION

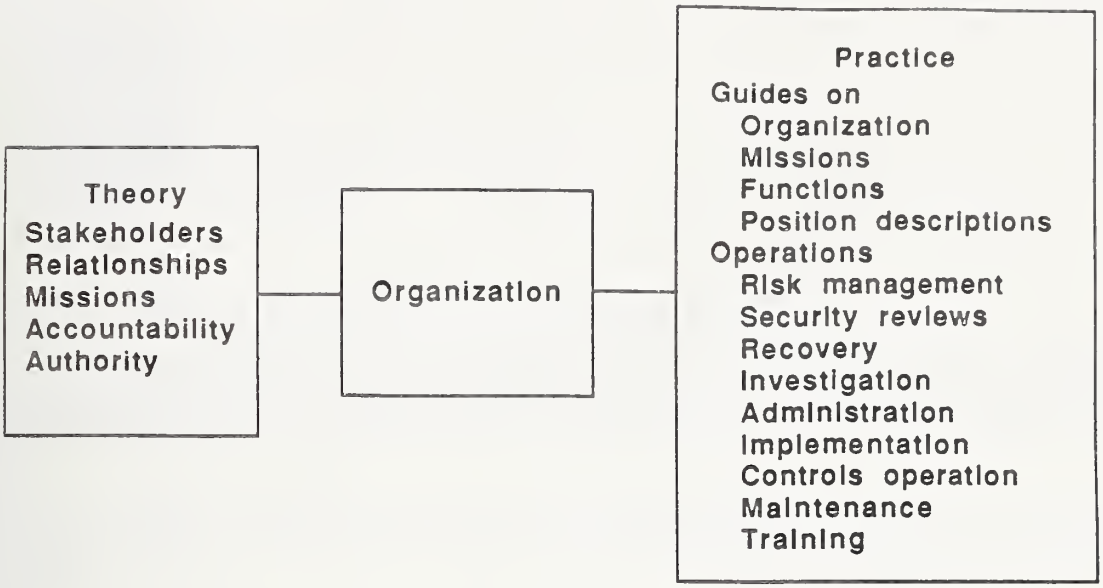
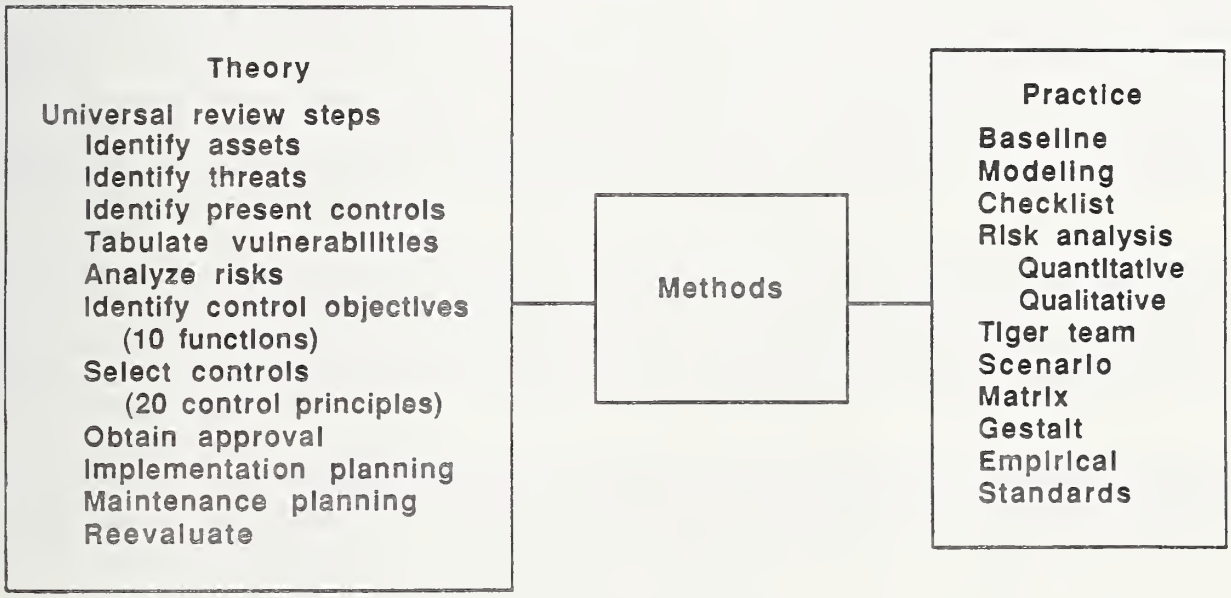


FIGURE 5 FRAMEWORK METHODS



INTEGRITY WORKSHOP - OUTLINE CONTRIBUTION

JTHOR: Tom A Parker
ICL Defence Systems
Eskdale Road
Winnersh
Wokingham
Berkshire RG11 5TT
England

My interest in this workshop stems from two distinct activities:

- design and development of a Bl secure version of ICL's proprietary Operating System - VME, and further planned enhancements to provide better support for a Clark-Wilson style policy.

- international standards work on secure distributed systems which has produced results which show how the ideas of Clark and Wilson generalise to distributed systems.

I shall cover each of these topics in turn.

VME support of Clark-Wilson

The VME system with its High Security Option supports a Mandatory Integrity Policy which partly but not fully supports C-W 'triples'. Under this policy users and the software they use to access data are both given an integrity category clearance. Write access is permitted only if the clearances together satisfy the Mandatory Policy check. ICL can demonstrate, (and does so on a regular basis to customers) how this approach can be used to control typical commercial transaction sequences like: accept-order, authorise, despatch, invoice, receive-payment, authorise-receipt; or in a program development environment: develop, handover, test, handover, live.

Extensions have been proposed and will be implemented over the next year which extend VME's Discretionary Policy to include explicit statements along the lines of: 'this user may access these resources via this application'. The policy is discretionary in the sense that it is not label based, but the proposed ACL triples can be centrally controlled by a Security Manager.

In our design of an integrity policy for we have drawn an implicit distinction between two kinds of lack of integrity (it would be interesting to speculate if it would be useful to define more). They are:

- correctly formatted but inappropriate (spoof data)

- incorrectly produced but appropriate (genuine unedited data)

An application can be written to defend itself against the last form of data input; input-edit programs are designed just to do that, but it is the job of the Operating System to provide the secure environment which defends against the first form of input.

On a more general point, it might be interesting to discuss why C-W decided to leave the enforcement of TP sequencing to the applications themselves (using CDI values). 'Well formed' sequencing could have been made an explicit feature of the model.

Distributed Systems

The European Computer Manufacturers Association (ECMA) has a group working on an Application Layer security framework for distributed systems. The group is called TC32/TG9; its work is well advanced and has produced some requirements on distributed systems which look very much like generalised C-W concepts.

The ECMA work has shown that access authorisation decisions, whatever the way in which the access control policy is expressed, are always made in the context of access privilege attributes possessed by the accessing subject and control attributes associated with the object being accessed. The nature of a distributed system demands that subject-related attributes be capable of being communicated via communications protocols, and much of the attention of TG9 is now being turned towards the standardisation of these and related protocols. It is also a common requirement in a distributed system for subjects to request other subjects (active software entities) to perform actions for them on their behalf. In some cases the intermediate subject requires no additional authority from the initiating subject to perform the action; in other cases additional authority to perform the action is required.

The figure below shows such a situation:

Subject A wishes to access an object X which is protected by an access control policy implemented in AC. Subject A is using Subject B as his/her/its agent for doing this. Both A and B may possess privilege attributes that are relevant to the access, and AC may require to be presented with the attribute both parties.

For example if A is a human user (or software representation of one) asking Print Server B to read file X from a File Server whose access control logic is AC, then AC may require evidence of both the requestor A's identity and, the security clearance of B (if the file is a Secret file the File Server may only release the file to a Print Server that has Secret clearance).

This two-subject picture, which together with the object looks remarkably like a C-W triple, generalises to any number of subject 'components', with the access control logic in the end application making its authorisation decision in the light of the privilege attributes of each subject component. Notice also that whereas the C-W model considers only the identities of the components involved, a more general model is one based upon their access privilege attributes (in the TG9 model a subject's identity is one such attribute).

Conclusion

Although this note skates over a number of basic concepts which are better described in the VME and TG9 documentation, I hope that the flavour of the work being undertaken in these areas has come across. Clearly I will be happy to present some or all of the above ideas in more (or even less!) detail at the workshop.

Tom Parker
10 June 1988

Invitational Workshop on Data Integrity
General Motors
Thomas R. Peltier

Data Integrity Issues

While discussing the scope and topic of this paper, a number of data integrity issues were identified. Although many issues were considered and continue to remain concerns, a consensus was reached on the importance of the following three issues:

1. Maintaining the accuracy of information in the integrated mega systems being developed today.

2. Failure of upper management to treat data as a crucial resource.

3. Excessive access to Corporate Data.

Maintaining the Accuracy of Information in Major Integrated Systems.

As systems get more integrated and therefore do more things, a number of data set owners may be present and the assignment of overall system ownership becomes more difficult. A simple change in the application programming can have a multiplier effect on other systems because of the increasing level of system integration. Because of this complexity, policies and procedures must be established to ensure that proper controls are built into an application at the start of system development. It is essential that the development of new information systems and major enhancements of existing systems be carefully managed. These are complex tasks involving many people over extended periods of time.

Management of these "mega" information systems can be made simpler if the controls are prioritized early in the development process. The GM System Development Process emphasizes the system user's responsibility for the development of the systems. Each system is seen as a joint undertaking by the user departments and the data processing entity, who work together and share responsibility. While this system of controls is effective in the development of new systems, a concerted effort must be made to retrofit these controls on existing systems.

Some of the concerns for both new and existing systems are:

1. Establishment of a system "owner" to ensure that changes or enhancements are implemented in a controlled environment:
2. Audit trails are sufficient to determine where, when and by whom changes, accesses or deletions to the system data or programs were made;
3. The data has been reviewed by the "owner" and proper classification has been assigned;
4. System, Corporate and legal requirements for the back up and storage of data offsite has been reviewed and is being met on a timely basis; and
5. A business resumption plan has been completed and coordinated with data processing.

Failure of Upper Management to Treat Data as a Crucial Resource

The growth of integrated data processing systems has happened so quickly that upper management often is not aware on its reliance on computer systems and the information they process. Daily decision making, as well as long term strategies, are based in part on computer generated information. The ability to conduct business in a manual environment is no longer a viable alternative. Without the ability to access information in a timely and effective manner, business efficiency will almost

immediately be halted and the on going survival of the business may be jeopardized by longer term disruptions.

Corporate information must be viewed by management as an asset just as important to the corporate well being as cash reserves. Upper management must support a level of controls associated with the true value of the information processed in the corporate activity.

Management has the ultimate responsibility to ensure that corporate data is properly protected from unauthorized access, modification, disclosure and/or destruction. There is a clear and present need for upper management to accept the extent business is dependent on computer data and take commensurate action to protect computer data and systems. There must emphasis on establishing policies and procedures and movement away from the "trusted employe" syndrome.

Excessive Access to Corporate Data

The complexity of access control administration in the current data processing environment is often too slow and cumbersome to allow the employees to complete their tasks in a timely and efficient manner. With the proliferation of users trying to access corporate data, the controls often suffer so that departments can "get the job done".

As remote access, networks and distributed processing make possible still greater access to computer generated data, the potential for unauthorized access to confidential and proprietary data increases. Therefore access control procedures are becoming more important. Even with the development of procedures to address security and business resumption issues, the question of continued adherence to controls and procedures remain. A proactive program informing and reminding the user community of their responsibilities is an important part of the over all security picture. This, coupled with the efforts of the Audit Staff, will increase the overall information security posture of the corporation.

Workshop Position Paper
Sig Porter

This note defines positions which I would undertake to assert or defend at the Integrity workshop. This is a response to Courtney's request (in his strawman paper[1]) for "constructive controversy". My principal object in sections 1 and 2 is to point out that the Courtney definition is too confining. In section 3, I briefly suggest that human factors can be profitably applied in integrity policy.

1. Avoiding A Single Definition of Integrity

Courtney's strawman suggests a definition for integrity. This is one more of the many meanings which have been proposed in the past [2]. In my opinion, effort to find a single definition is misdirected. It is, however, of great value to know what is meant when various authors use the term integrity. Consequently, we should not focus our attention on a single definition of a word, and then restrict our attention to the concepts in that definition. We should, instead, look at the various useful concepts which different people associate with that word. Only then will we be equipped to consider what word (or words) most felicitously expresses those concepts.

Courtney puzzles about what Clark & Wilson [3] meant by integrity in their paper. It seems unlikely that Clark & Wilson would settle (given a choice) for using their mechanisms to insure, for example, that as a result of a Transformation Procedure (TP), 90 percent of the data in a Constrained Data Item (CDI) is correct. Thus, contrary to what Courtney would apparently like to assume, Clark & Wilson did not mean a firm knowledge of the failings in quality of data.

For reasonable commercial utility, integrity in a C&W CDI must be the assurance that the CDI is the result of a traceable authorized action (Transformation Procedure) on input data (from an identified and authenticated source) and pre-existing CDIs. Note that the integrity of the result (output CDI) depends on the integrity (by various appropriate definitions) of the input and process:

1. The TP has integrity. (It is certified to perform the required function and only the required function).
2. The system has integrity. (Only an authorized user is able to execute the TP. CDIs are altered only as a result of TPs.)
3. The input data (UDI) has integrity. (It is known to be from the claimed source, and is the same as was entered by that source.)
4. The input data (CDI) has integrity. (It results from the process which we are now recursively defining.

2. Levels of Integrity

Courtney and others view integrity as an absolute state with no levels or degrees. While I do not claim that this is never valid, I feel that some useful definitions of integrity inherently have a measure, and hence degrees, and possibly levels. Rather than discussing "knowing" or preventing", I would talk about "knowing with some degree of certainty", or "preventing with some probability". I regard these probabilities as measures of integrity. (It is not clear what Courtney means by "There are no degrees of integrity. ... Integrity reflects trust... There can, however, be degrees of trust.")

The field of communication provides useful examples in this area: When we receive a message, we are concerned about its integrity. Here the integrity is generally viewed as assurance (or probability) that the message is the one originally sent and is from the claimed source. (We do not talk about "degree of assurance that a message has integrity" (or "degree of assurance that only half the characters in the message have been altered"). Cryptography and error correcting/detecting codes provide frequently applied integrity assurance tools. These tools permit the system designer to specify parameters (e.g., key length) which will determine probabilities that specified assurances are provided.

The field of security is not enhanced by defining away the probabilistic aspects of integrity.

3. The Application of Human Factors to Security Design

In this section I would like to suggest an approach to security which I believe will be helpful for integrity. This approach is based on concerns about (1) the effect of the Orange Book on the security art, and (2) the common view that security is incompatible with a comfortable user environment.

I believe that the OB, by converting the best techniques of the late 70s and early 80s to theology, has actually slowed the development of the security art. Too much time and talent is spent interpreting the OB and trying to accommodate its mechanisms. Much of this time and talent would be more productively spent developing new approaches to security. I am concerned that some people may want to approach integrity by defining a sort of Orange Book for Integrity. I would prefer to see, as will be discussed below, a finer grained policy than found in the OB.

Most people view security as interfering with a comfortable mode of working. An example of this can be seen in the reaction to the recent Internet virus, namely concern that if the (integrity) holes are plugged, the net will not be as usable.

Solutions can often be found by considering security as a human factors problem [4]. An example can be found in [5], where consideration is given to what users actually need and want in their normal activities. When such consideration is given, security can be achieved without undue inconvenience to the user. Based on these needs, a security policy with finer granularity of control was proposed to better control Trojan horses without imposition of MAC.

[5] also includes policy proposals which are intended for protection against viruses, and hence in support of both system and data integrity. (The policy rules are explicitly present, but not discussed in [5].) These proposals require a program to have explicitly granted (by hand) privilege to create (or alter) an executable program. This does not inconvenience the user (except slightly, perhaps, for compiler developers), since most programs do not require this privilege. The few programs which reside in user space and require the privilege can have it specially applied. Obviously, users will not grant the privilege to untrusted imported programs, such as games and editors.

---

1 Courtney, R., "Some Informal Comments about Integrity and the Integrity Workshop", undated.

2 Porter, S., and Arnold, T., "On The Integrity Problem", 11th National Computer Security Conference, 1985.

3 Clark, D., and Wilson, D., "A Comparison of Commercial and Military Computer Security Policies", 1987 IEEE Symposium on Security and Privacy.

4 Porter, S., "A Password Extension for Improved Human Factors", Crypto 81 and Computers & Security, Vol. 1 No. 1, 1982, North Holland Press

5 Porter, S., "Dac Not Inherently Flawed", Dockmaster Criteria Forum, Transaction 1194, 10/26/88

Why Integrity is Important to Me: A Position Paper for the Invitational Workshop on Data Integrity

Marvin Schaefer

Trusted Information Systems, Inc.
Glenwood, MD 21738

It has been a tenet of traditional computer security that the "TCB boundary" forms a partition between security-relevant and non-security-relevant code. The non-security-relevant code is often called "untrusted" code, in the sense that the properties of untrusted code should not be capable of subverting the security policy implemented by the TCB.[1] Recently, discussions have raised questions as to whether such a TCB boundary could be made to exist for integrity-critical applications.

I believe that with the exception of certain simplistic label-based definitions of integrity,[2] the concept of an integrity TCB is too difficult to implement. That is, I believe that in the general case, it is not possible to implement a mechanism that will either (a) guarantee the legitimacy of all data updates derived from data introduced into a system by users or user-written programs; or, (b) guarantee the legitimacy of all data updates derived from data introduced into a system only by authorised sources or authorised programs, if the update transactions are driven by user commands or user-written programs. Alternative (b) represents an extremely pessimistic hypothesis, for it claims that even if a set of data were ever placed in a legitimate or consistent state it is doubtful that even the drastic measure of restricting system operation to only a small set of authorised system state changes and the introduction of no new external data would prevent subsequent corruption of the database.

My more recent professional activities have identified me as a member of the computer security community. However, the path that led me into computer security touched on the technologies and practices of languages and compilers, operating systems, database management systems and formal verification. Those early experiences really emphasised the fundamental importance of considering integrity as a data correctness issue.

There was a time when the meaning of 'integrity' was closely identified with the concept of 'computer security'. This can be seen by reading classics like Don Parker's Computers and Crime. There was concern over the unauthorised manipulation or modification of data, especially for purposes of fraud. It was also recognised that people who made unauthorised use of computing resources often did so either by exploiting an error in the operating system or by corrupting or manipulating the data relied upon by the operating system for its correct operation. The latter form of attack would employ or exploit a smaller 'hole' or vulnerability (generally not considered important to the system's developers) as a means to get the still 'correctly functioning' operating

---

[1]When the term "security kernel" was used in lieu of TCB, it was contended that the kernel should be capable of protecting the system from security violations, even if the adversary were to write all of the untrusted code permitted to run on the system.

[2]'Simplistic' ought not to be read as a pejorative in this context. In many applications, there is value in being able to determine that data has not been modified since it was created and sealed with a cryptographic label. In other cases, modeled by Biba, Lee, and Shockley, it is assumed that the value or correctness (integrity) of data can be preserved or improved only by a few very special classes of program or data, that all other modifications will lower its value, and that a lattice-based system of labels can be used as the basis for integrity-preserving access control mechanisms.

system code to perform new acts on behalf of the interloper.

Though few would have classified it as such in the early days, the act of systematically feeding bad data to an operating system (or of changing good data to bad) is arguably a necessary part of the penetration scenario. The generated bad data images may be within the computer's primary or secondary store, and may include portions of the control data (central tables) or could take the form of modified or new instructions.[3] Modifications to the system's memory may take effect immediately, while changes to data in secondary store may not subvert the system until the next time it is generated or booted.

The DoD Trusted Computer System Evaluation Criteria (TCSEC) does not provide an explicit enumeration of the many integrity requirements that are implicitly assumed necessary for a trusted system. The TCSEC requirements suggest that the TCB and its data must be implemented reasonably correctly, that the TCB code and internal data must be protected and kept reasonably correct and consistent, and that unauthorised users and subjects ought not be capable of penetrating the system or modifying key protection-critical data.[4] The TCSEC integrity requirements need to be reexamined in light of current understandings and presented in a more unified form.

Database administrators often observe that data, once corrupted, tends to propagate additional data corruption throughout a database. This is because new data is often generated as a function of existing data values (often by copying some existing data into new data records), or data records to be modified are selected based on the values of existing data that is not to be changed. The rapidity with which "dirty data" can beget further dirty data is the basis for the legendary saw: "Garbage in, Garbage out".[5] It is so difficult to prevent the generation or incorporation of dirty data into a database because it is generally impossible to produce an algorithm that differentiates between legitimate and illegitimate data values.

Systems often crash following a penetration or penetration attempt. Sometimes the crash is caused consciously by the interloper: a secondary image of system data is modified and it is necessary to the penetration for the secondary version of the system to become the primary version.

A system may also crash because of an inconsistency in its internal data resulting from an incomplete modification to the central data. Such crashes may even result because the system read and used some partially modified data as the basis for state changes even while the interloper was in the process of modifying other data values. [System penetrators do not often have the luxury of

---

[3]It is consistent to classify executable code as data, since von Neumann architectures are machines that execute [some of] their data. Necessarily, interlopers would consider microcode as data.

[4]The TCSEC requirements include: [C1] the need for "features ... that can be used to periodically validate the correct operation of the ... elements of the TCB"; [C1] the need for "a domain for its own execution that protects it [and its data structures] from external interference or tampering"; [B1] the need to eliminate flaws that would "permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy"; [C2] the need to prohibit "unauthorised access to the audit or authentication data"; [B2] the need for the TCB to be "relatively resistent to penetration"; and [B3] the need for "procedures and/or mechanisms ... to assure that ... recovery without a protection compromise is obtained".

[5]A computer virus can be thought of as code that is introduced into another body of code such that its action is not consistent with the specification of the latter body of code. As such, its properties are not significantly different from the general dirty data propagation problem. If a body of code reads (executes) an executable virus, the [data value of the] virus may result in the corruption of the process that executes the virus as well as any data structure modified by that corrupted process, including other executable data structures.

modifying data as though with a DBMS that can lock a transaction as an "atomic" act!] In such cases, the very act of attempted penetration may succumb to the side effects of dirty data propagation.

This would suggest that systems should take strong measures to prevent the unvalidated modification of their own central data structures. Parnas, Hoare, Dijkstra and Horning have all advocated disciplines of mutual suspicion in system design and implementation so that data values received as parameters are checked for validity (or reasonableness) prior to being used, even at the potential for a significant drop is system efficiency. While some forms of validity check are easily characterised (e.g., is the variable within its acceptable range of values?), it is rare that it can be determined that a reasonable value is also a correct value. Given that a system calamity can occur even if only there are two different concurrent values for the number of active processes on the machine at some time, the correctness problem must be considered significant.

Of course, much attention must go into the problem of protecting a system and its users from potentially malicious users. While this is a valid concern, the experiences of database administrators with the dirty data problem has shown that concern need also be accorded to the problem of honest users (who may misinterpret the meaning of some data) using honestly conceived, but incorrect, applications programs.

IMPORTANCE OF MANDATORY INTEGRITY CONTROLS

W.R.Shockley
Gemini Computers, Inc.
Box 222417, Carmel, CA 93922

Recent initiatives in the commercial data integrity arena (such as this workshop) seem entirely appropriate in the context of our nation's increasing dependence upon automated information systems, their increasing interconnectivity, and the continuing proliferation of malicious and and mischievous attacks upon I question, however, the proposition that the single most appropriate response to these threats is the definition of a long-term research agenda while currently available measures for the abatement of the threat remain unexploited. I am speaking specifically to the definition, by end-user organizations, of appropriate mandatory integrity policies so that currently available technology can be applied to their enforcement by their suppliers.

Traditionally, the attainment of computer security rests upon a triad: policy, mechanism, and assurance. Sound enforcement mechanisms, and the techniques for assuring that they work, (that is, that the associated policy is enforced) have been with us now for a long time for so-called mandatory integrity policies. Such a system of controls has, however, been consistently rejected by commercial policy makers (or their advisors). The usual arguments advanced are that 1) they don't actually ensure correctness of data, and/or 2) they are inconsistent (or irrelevent) to commercial needs.

The first argument is intellectually equivalent to arguing that it isn't useful to fence a cornfield because the fence won't help the corn grow. The essence of mandatory integrity is that it allows one to define who the outsiders are (even within your own organization) and keeps them out of the critical data. Although this doesn't ensure the correctness of data, it certainly limits the threat of how the data might come to be incorrect.

The second argument has never, to my knowledge, been successfully maintained once an adequately clear counter-example has been proposed. Generally, what is shown is that some subset of a full mandatory policy (e.g., levels alone) is inconsistent with the counterexample. What is being claimed is that if the commercial policy itself is sound, the "mandatory part" of the policy can be identified -- and there are significant advantages in assurance available for enforcing this part seperately. As an example of what can

be done, Steve Lipner's paper on "Non-Discretionary Controls for Commercial Applications" (1982 IEEE Symposium on Security and Privacy) is of interest as a carefully worked-out mapping.

The advantage of explicitly recognizing the mandatory integrity component of a commercial policy, is that much stronger assurances (essentially, a closed technical demonstration) can be provided that a particular system unequivocally enforces the policy (a stronger statement than that it implements some mechanism correctly.) This is one of the few arenas where a body of techniques exist to demonstrate the correctness of the specification, as opposed to the correctness of an implementation against possibly incorrect specifications.

The most appropriate immediate step for organizations to take is simply to define an appropriate mandatory policy for labeling organizational data, defining the circumstances for clearing users to modify the data, and defining the circumstances for certifying procedures (e.g., manual procedures) for making changes to the data. Such a policy is meaningful even without the utilization of automated systems for enforcing it, and serves as the essential prerequisite for establishing what is required of a new automated system (e.g., will it process data of different integrity? be accessible by users with different clearances?) so that risks and appropriate countermeasures can be selected for new acquisitions. Among the countermeasures that can be considered is the use of a trusted system, capable of enforcing the policy internally.

The measurable benefits of such systems are that they are demonstrably robust in the face of Trojan Horse and virus attacks -- in the sense that, while they remain vulnerable to "infection" (as a system), when the infection is eliminated, the protected data stands a good chance of being found intact. The only way for a newly-arrived virus to modify protected data is to attach itself to an uncertified program in such a way that it slips through the organization's program certification process undetected -- a process that can be made as stringent in its requirements as the organization may wish to make it. Discretionary-only systems can make no such claim.

DEFINITION AND CONCEPTS OF DATA INTEGRITY

by

Stelio Thompson-Sittas

INTRODUCTION

The concepts and definitions of data integrity developed by the working group and as articulated by Robert Courtney and Dr. Willis Ware, are thought-provoking. Therefore, I would like to propose the following alternative concept of data integrity.

DEFINITION

The ability of an element, or system of elements, (whether data, programs, hardware, personnel, etc.) to retain its attributes, and associations and guard against non-intended changes as the element proceeds through its lifecycle.

DISCUSSION

The integrity of an element can be considered on a continuous scale, ranging from extremely high to extremely low retentive ability (At the risk of offending the gods, I would like to state that integrity is not a dichotomous, binary function!) To follow Dr. Ware's example of the holes in the boat and cracks in the airplane, it can be demonstrated that the body of an airplane always has cracks present, even though some cracks are so small they can only be detected by use of precision instruments. Likewise, every boat has some leaks, however, both the cracks and the leaks are of an acceptable level which allow the boat and plane to be considered "safe" for their intended use, and can be considered to have integrity.

Additional, incremental levels of integrity can be obtained in most situations. However, the costs associated with these increasing levels of integrity could increase at such high levels that the additional integrity level would become prohibitive.

In relation to any element or data integrity, the term "trust" could be used only as a common use synonym to express degrees of integrity. That is to say, an element with high levels of integrity, where the attributes are retained under any condition, and could be trusted to be as expected (a priori). The term "quality" is generally used to express the relevance, fitness for use, or confidence level of an element. Generally, quality is a design function or a result of a low integrity process.

The multiple attributes associated with an element generally requires a combination of safeguards to ensure its ability to retain its properties and associations and guard against non-intended changes. The contribution level of each of the corresponding safeguards combine to produce the total level of integrity. This also supports the concept that integrity is not binary.

The authors of the Orange Book (DoD Trusted Computer System Evaluation Criteria, 15 Aug 83) provided some excellent guidelines for computer security, and in fact, also treat "data integrity" as a continuous process with differing degrees.

The Department of Defense, over the years, having varied needs for security has provided a great deal of definition and common use language dealing with security concepts. In reworking old definitions, and addressing new areas, we need to take advantage of the precedent use of terms.

SENSITIVITY

A relationship can be established between integrity and sensitivity in the following manner:

* Unauthorized access to sensitive elements will result in high impact and high losses.

* One of the attributes of an element or combination of elements (system) is the ability to control access to only defined and authorized subjects to prevent unauthorized disclosure.

* The losses that can be attributed to the failure of the controls and the resultant disclosure of the sensitive element is disproportionately high due to its overall grave effect.

* A higher level of integrity of the element or the process and its associated higher cost, is justified as a result of the disproportionately high estimated losses.

For example, it is clear that the higher the integrity of the control access system or process, the smaller the likelihood that sensitive elements will be disclosed resulting in lower potential losses.

Integrity is related to sensitivity whenever we consider the access attributes of integrity in any element, that could result in disclosure.

The Swedish National Audit Bureau
Audit Department 3
Section for Information Processing

Peter Nilsson
Eva Sparr

DATA OR INFORMATION QUALITY

The following report from the RRV (the Swedish National Audit Bureau) forms the basis for a proposal for the standardization of the terms used in information quality in Sweden. The terms and the model are to be published in a technical report to be issued by the Swedish Standards Association (SIS). It has also been proposed that these terms and their definitions are to be included, where necessary, in the next edition of their Data Processing Glossary (SS 01 16 01).

1. Introduction

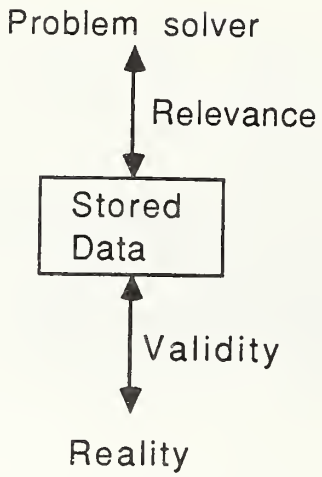
Quality is a matter of characteristics. The quality of received information is a function of its characteristics. It cannot be stated that certain information is better than other information. The perception of quality depends upon the purpose for which the information is to be used.

The starting point is that all information which is collected is always related to a question or a problem. Thus, the process of data collection is goal-oriented. The system acts as a channel between the process of collection and the use of this data. The perspective is the user who needs information to solve a problem.

The most relevant term to use in this context is information quality. Seen against the definitions of data and information, the term data quality covers the quality of the processing of the data in the data processing system. The quality of information concerns the quality of the content of the information up to the point where it reaches the user, and includes the user's interpretation and application of the data received. Thus, data quality is a subset of information quality. However, as it is used today, data quality has the same meaning as information quality. Accordingly, the terms "data quality" and "information quality" will be regarded as synonymous.

2. The Model

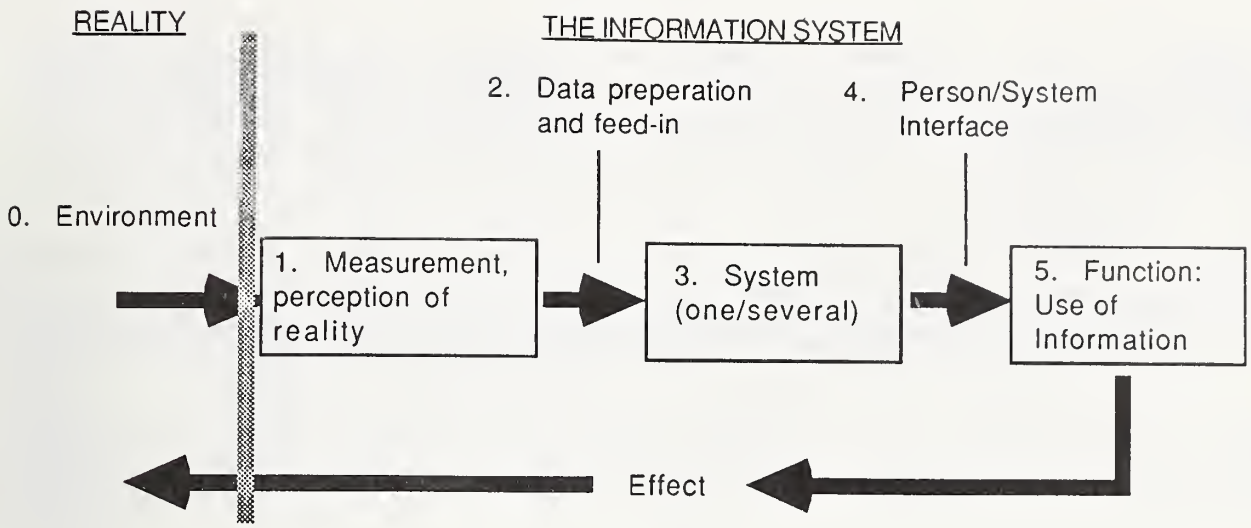
In principle, there are two aspects to information quality: they are relevance and validity.



Relevance describes whether stored data is appropriate, i.e., whether it is relevant to the problem which the user wishes to solve. Before he can determine whether information is relevant the user needs to know, for example, what the data involves, under what circumstances they were gathered and the number and type of errors the data contains. The term validity describes whether the stored data are "true", whether they are an accurate description of reality. Arriving at a measurement of validity involves making a comparison between stored data and some reference material which gives a better description of reality. This may be a comparison based on data gathered from forms, a direct comparison with reality, or from corresponding data stored in another register (which is considered to be of better quality).

Data may be valid but not relevant, i.e., they may not be appropriate to the problem which the user wishes to solve. Moreover, there may be shortcomings in validity, for example, ten percent of the information is wrong, but in a given context this causes no problems. Error-free data may not be required in order to solve the given problem. In this case, the information is relevant. The problem definition is what determines the information is relevant and thus what degree of validity is required.

Below is a description of the information flow from the point at which the information was collected, to the user, and the effect which occurs when the user applies the information received in his activities. Errors may occur at all stages in the flow from the information collecting stage to the user. Too large errors will distort the content of the information, which may have negative results.



0: Environment, or that which one wishes to transfer certain information about.

1: One selects, estimates and measures that part of the environment which is appropriate to the problem that must be solved; problem definition. Descriptive parameters may be: coverage, accuracy and timeliness.

2: The interface between collecting information and the system. Collected information is coded classified, aggregated etc., to adapt it to the system.

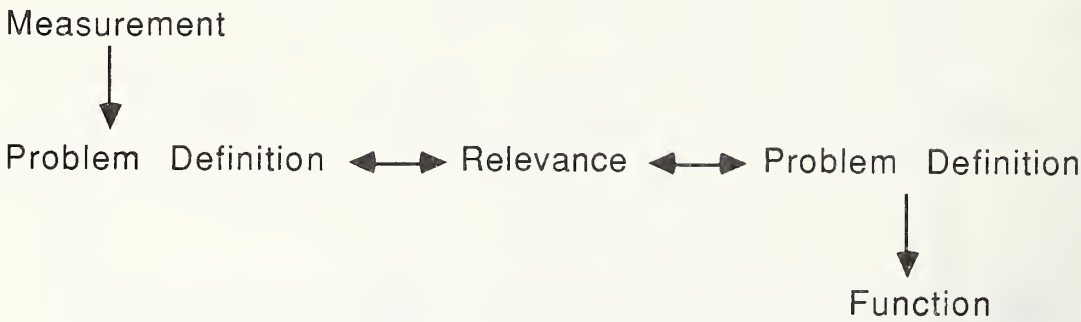
3: The data processing system. A collection of hardware units, methods and procedures and, possibly, even people. Stored data must be valid.

4: The interface between the user and the system. Data must be physically and logically accessible (retrieving the data and the data retrieved being susceptible to interpretation). It may also be desirable to have other functions which the user may require to control and check information received.

5: The function in an organization or agency. Information received is used in a variety of activities, here called problem solving. In order to find out which information is required, the problems must be analyzed and specified in a problem definition, and a precise definition of the information required then set out in a needs specification. The needs specification, which should be documented, may contain details on what subjects and characteristics are desired, the degree of detail with which reality is to be described, when data are to be received and how old they may be, and how many errors the data is permitted to contain (coverage, accuracy, timeliness and error tolerance). Problem solving which is dependent on information received has some effect: internal or external, positive or negative, expected or unexpected.

0 compared to 1: Problem definition. Information is always collected to provide answers to problems.

1 compared to 5: Relevance. Compare the collectors' and the functions' problem definitions. Information which is collected must correspond to the question which the function wishes to answer.



1 to 5: Errors may occur in the entire communication process from measurement to user (1, 2, 3, 4 and 5). The transfer must be objectively correct.

2 to 4: Correct data processing. The data fed into the system must be processed according to the specification. The data processing must be objectively correct.

3. Definitions

Some of the definitions below are already published in the current version of the Swedish Standards Association (SIS) Data Processing Glossary.

INFORMATION QUALITY: The degree to which information is usable to a given user and for a given problem, including the effect on the user and the environment.

INFORMATION: The content of data.

DATA: The representation of facts, terms, or instructions in a suitable form for transfer, interpretation or processing by people or automatic aids.

SYSTEM: A collection of hardware units, methods and procedures (manual or automatic) and possible also people, organized to perform data processing.

FUNCTION: A role in an organization. A user solves problems according to his/her function. The system should provide the information required. The function specifies the user's problem definition.

OBJECTIVE CORRECTNESS: That no errors occur to affect the quality of the information. Errors may occur anywhere in the flow from the information collecting point to the user. Errors may systematic or unsystematic. Systematic errors are errors caused by one or more factors whose effects obey some law or follow some pattern. Repeated measurement will cause the same systematic error to occur. Unsystematic errors are caused by chance, negligence and the like.

PROBLEM DEFINITION: Specifies the problem to be solved and thus the quality of information needed. The needs specification for the information may be expressed in terms such as accuracy, timeliness, coverage and error tolerance. The terms used must be unambiguous.

RELEVANCE: A is relevant to B if A's problem definition agrees with B's. Knowledge of the problem definition is essential before the relevance of given information can be assessed. It is appropriate to give details of the relevant timeliness, accuracy, coverage and the number of errors that can be tolerated before the information is irrelevant.

COVERAGE: That all relevant objects with characteristics are fed into the system. Loss is a subset of coverage. When a characteristic or variable is missing from an object existing in the system.

ACCURACY: How well the specified characteristics are measured are given a value. Measurement is done with certain accuracy on a scale (according to the nominal, ordinal, interval or quota-scale), which may be more or less detailed.

ERROR TOLERANCE: A measurement of the number of errors that can be tolerated before the information is irrelevant.

STABILITY: The stability of a variable when repeated measurements are taken.

DEGREE OF DETAIL: The amount of resolution in describing reality. May be expressed as, for example, the number of classes which describe the phenomenon. The more classes there are, the greater the degree of detail. The degree of detail is in fact periodicity. A scale with a high degree of detail, e.g. many classes, involves a higher risk of error.

PSEUDO PRECISION: A degree of detail which is too high in relation to the stability. Gives an impression of greater precision than is appropriate.

TIMELINESS: The time aspects of the information. Timely data is data which reflects the point in time which the user wants information on. This may be the latest, most current data, or data on a given point in the past.

TIME OF EVENT: The time at which the event one wishes to measure occurred. Data relates to that time.

TIME OF MEASUREMENT: The time on which the actual measurement is based e.g. a period of measurement of a week to represent the whole year of the collecting of information after an event has occurred. The time of measurement may be the same as the time of event.

TIME OF USAGE: The time at which the information is used. The difference between the time of event and the time of usage is the age of the data.

UPDATING: Modifying a volume of data with current data according to a specified procedure. Initiated by changes in time or in one of several variables specified in the degree of detail.

VALIDITY: The degree to which the data reflects reality. A weakness in, for example, updating, can result in less validity.

RELIABILITY: A measurement of the degree with which the system supplies the information of a given quality which it is said to deliver.

ACCESSIBILITY: That the user has access to the information. The interface between system and user.

RETRIEVABILITY: The system produces relevant data. A measurement of the search function in the system.

INTERPRETABILITY: That data received can be interpreted, that they have a structure which gives the user the correct information.

OBSERVABILITY: The possibility of observing the behaviour of the system and its interaction with other systems.

MEASURABILITY: The possibility of measuring/checking a variable, component or function in a system. Information is received on its status but this does not mean that it can be influenced.

CONTROLLABILITY: The possibility of controlling or changing a variable, component or function as desired.

CONSISTENCY: The degree to which data stored in a system agrees or are consistent. Data may be objectively correct but logically inconsistent.

COMMENTS ON INFORMATION INTEGRITY ISSUES
for the NIST-hosted 1989 Invitational Workshop on Data Integrity,
January 25-27, Gaithersburg, MD

John M. Thurlow, Exxon Central Services

Disclaimer

The following discussion of priorities for resolving various aspects of managing information integrity reflect my opinions, and not necessarily those of Exxon management, but the discussion is derived from issues present in a modern commercial organization.

Primary Issues

For me, the most important issue is improving the appropriate use of information systems technology on behalf of the objectives of a commercial concern. In this general arena, the two aspects of the problem are: 1) providing simple concepts of integrity for not only those who specify, design, and deliver applications of I.S. technology but the end-users of those applications as well, and 2) providing a clear statement of requirements to the suppliers of the underlying IS technology.

For those concerned with applications delivery and usage, the concepts need to be augmented by general approaches for understanding the value of information integrity in specific circumstances, and achieving the appropriate level of integrity. That is, they need systematic approaches to identifying information integrity needs and the benefits of corresponding levels of integrity to balance against the costs of an effective approach to achieving those levels.

Because improved organizational productivity is also a requirement, users of IS technology will seek out suppliers who have recognized the needs to support information integrity, and provide suitable products. Application deliverers who use more compliant technology will be more productive by the amount of work which they do not have to provide, because required facilities have been provided by the supplier.

In a typical commercial concern, one can identify a considerable proportion of the efforts to introduce and maintain IS applications which is devoted to ensuring the requisite information integrity in spite of shortcomings of the underlying technology. A good example is found in common systems for delivering electronic mail to users in an environment where multiple nodes are connected by telecommunications links. Vendor-supplied systems, including the supporting operating systems, typically are not explicitly designed to ensure the mail is delivered to the intended recipient, and is not lost or delayed to point of being effectively lost. Additional work, both one-time and ongoing, is often needed to provide the required level of int grity (no lost mail).

"Integrity" and "quality"

I would like to move away from using "quality" as a term having a parallel meaning to integrity in this context. For converts to quality management thinking (Deming, Crosby, et al.), "quality" means "meeting requirements". The appropriate level of information integrity is one of the requirements to be met in an application of I.S. technology. Other security-related requirements could include appropriate availability and confidentiality. It is helpful in gaining attention to integrity issues to keep the terminology simple and consistent with general usage.

Models

Models are useful aids in establishing issues for attention and a common language for discussing them. For identifying information integrity requirements, the structure of the Clark-Wilson model seems to be a very good starting place. It is relevant to the interests and experience of a commercial organization, and recognizes the role of humans in information management processes in its discussion of segregation of duties.

I believe that the Clark-Wilson model merits the further thought that is being applied to it, and can be the basis for systematic processes designed to elicit information integrity requirements and approaches to meeting them. I think that as more practitioners become familiar with the principles of the model, and it is translated into less-rigorous terminology, it will be applied to great advantage.

3/31/89

Doug Varney, Kansas State University
Position Paper on Data Integrity

One of the most difficult aspect of the Clark and Wilson model for data integrity is the certification of Transformation Procedures (TPs). Since there is probably not a great desire to rewrite application programs that make it easier to certify we are left with the daunting task of certifying existing application programs. Compounding our problem is that for much software the source code is not available.

The certification process can be broken into two areas: that the correct data is being modified and that that data is modified in a manner that preserves its integrity.

Correct Data: Most of this can be pushed off on the access control triple to enforce that only correct data can be modified. Obviously this depends on the granularity of the CDIs in the access control triple, but it is a good starting place.

Correct Modification: Unfortunately program proving is not going to provide a silver bullet for proving that only correct modifications can occur. Without program proving we are left with testing and code walk throughs as the means of certification.

Testing: A testing criteria and method should be chosen to correspond to the level of integrity desired. C0 or C1 should be considered the minimum while for higher levels of integrity further methods such as revealing subdomains, boundary testing or C-reliable testing should be chosen. The amount of testing may also be influenced by the past history of the program. If there is a close relationship with the developer then access to the developer's testing data would be invaluable. Obviously this is easiest with the specification of new programs.

A close relationship with the developer does not exist in every case (shrink wrapped software for example). In the cases where certifier does not have access to the source code he can either trust that the developer has certified that program works according to specification or can disassemble the code and test that. Neither of these options are very desirable. The choice rests on the level of integrity desired for the system and the reputation of the developer.

Code Walk Throughs are also an essential part of certification. Preferably done in conjunction with the developer. Unusual programming practices should be questioned extensively. This assumes that compilers maintain integrity, which may not be a valid assumption in some cases.

There is a further question whether integrity can be maintained in a programming environment without great programmer inconvenience. The high rate of introduction of new objects makes maintaining integrity difficult. There is a threat of viruses. A method of such as a knowledge based name checker [Karger] and/or protection of executables by a manipulation detection code [Pozzo and Gray] is a possibility to limit viruses. Once in a nonprogramming environment the access control triple should be able to limit the spread of viruses.

Conclusion: Certifying Transformation Procedures and testing are closely linked. Certifying TPs is easier the more test data that is already accessible. The lack of source code makes certification very difficult. Special care should be taken in fluid environments to prevent the infection by viruses.

Integrity and Information Protection

S. R. Welke
W. T. Mayfield
J. E. Roskos

Institute for Defense Analyses

This paper is a response to the Strawman document "Some Informal Comments About Integrity and the Integrity Workshop" by Robert H. Courtney from the perspective of the Institute for Defense Analyses (IDA). IDA is currently being funded by the National Computer Security Center to analyze integrity and availability issues in tactical/embedded computer systems. This project is an outgrowth of requirements set forth in Department of Defense Directive 5200.28, Security Requirements for Automated Information Systems. The directive states that "safeguards shall be applied so that information retains its content integrity ....information shall be safeguarded against tampering, fraud, misappropriation, misuse, loss, and destruction."

IDA's framework for information protection uses the triparite set of integrity, confidentiality, and availability. These three aspects are inexorably linked; we can not look at one without considering all three. The "Orange Book" is particularly remiss in addressing only confidentiality issues and essentially ignoring integrity and availability. Many computer system protection mechanisms exist which support all three aspects, but these mechanisms have been primarily touted as supporting confidentiality.

For our purposes, data integrity is defined in directive 5200.28 as follows:

> Data integrity - the state that exists when data is unchanged from its source and has not been accidentally or maliciously modified, altered, or destroyed.

This definition reinforces the view that integrity is a binary property. Integrity is concerned with damage to data by actions that are malicious, fraudulent, accidential, or environmentally induced. These actions can be either internally or externally initiated.

We agree with the Countney's Strawman paper on several points.

- Integrity is binary; either you have it or you don't. Confidentiality and availability are also binary properties.
- The three aspects are closely linked to one another.
- Integrity is based on expectations of specified attributes. These expectations must be identified before any evaluation of integrity can be made.
- Trust is a relative measure that can be used to convey the degree of confidence one has in the safeguards applied to achieve data integrity, confidentiality, and availability. It is used in this manner in the Orange Book with respect to confidentiality.
- There are mechanisms upon which integrity, confidentiality, and availability safeguards can be implemented; some are unique to one of the three aspects, while others are shared between them.

We differ with the Strawman paper on what the point of focus should be at the upcoming Workshop. Courtney suggests that the next point of focus should be to attempt to "redefine" data quality. Quality is a primitive concept that has been shown to be undefinable, so attempting to redefine it might be a frustrating task. Instead, we recommend using the seemingly acceptable terms "consistency", "accuracy:, and "timeliness" to examine the possible contributions of known protection, reliability, and recovery mechanisms. By consistency, we mean that data is unchanged and agrees with itself internally; by accuracy, we mean that data is precise within specified tolerances; by timeliness, we mean that data is available for its intended use within specified periods.

Ensuring integrity requires dealing with data structures, data contents, process and user authorizations and authentications, and data constraints. Many protection mechanisms that are currently used to achieve confidentiality were originally designed to achieve integrity. We should examine the focus of each type of protection mechanism and ascertain its role in providing for one or more aspects of information protection. Many existing mechanisms focus on all three aspects of protection (e.g., control of access, damage prevention/detection/recovery). Other mechanisms focus on both integrity and confidentiality (e.g., separation of processes, separation by sensitivities or other categorizations). Still other mechanisms focus on integrity and availability (e.g., denial prevention/detection/recovery). These machanisms should be discussed to determine what they provide, and what is still needed.

APPENDIX C

The Call for Papers

CALL FOR PAPERS

Invitational Workshop on Data Integrity

January 25-27, 1989

Sponsored by and Held at

National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, Maryland

The National Institute of Standards and Technology is sponsoring an Invitational Workshop on Data Integrity to be held at NIST in Gaithersburg, Maryland on January 25-27, 1989. The Workshop will focus on the concepts of data integrity and data quality, and the characteristics, metrics, and principles needed to define and provide a suitable framework for data integrity and data quality.

This invitational workshop is a follow-on to the October 27-29, 1987 invitational Workshop on Integrity and Privacy in Computer Information Systems (WIPCIS). The latter originated as a response to a paper by Clark and Wilson presented at the IEEE Security and Privacy Conference in April, 1987. That paper compared commercial and military computer security policies. The 1987 Workshop focused on commercial sector interpretations of the issues of Assurance, Granularity and Function, Identity Verification, Auditing, and System Correspondence to Reality and related these to a data integrity model.

A subsequently-formed data integrity working group of the NIST Computer and Telecommunications Security (CTS) Council has proposed definitions for data integrity and data quality. These have been incorporated, along with other conclusions, in a paper written by the working group chair Robert H. Courtney, Jr. It is intended that this paper serve as a strawman to stimulate responses in the form of papers to be given at the January Workshop. The Clark & Wilson paper would form an example within this framework. Although computer and telecommunications system integrity is broader than data integrity, consensus findings about data integrity would contribute significantly to our understanding and handling of the broader concerns of integrity.

Papers are being sought from the computer security community for presentation at the Workshop. Papers could address but need not be limited to the following topics:

- Key principles for achieving data integrity.
- Key principles for achieving data Quality.
- The application of principles to achieve integrity and quality of data.
- The attributes of data quality (other than accuracy, timeliness and completeness).
- Is confidentiality an attribute of data quality?

- o Connection between Quality Assurance and data integrity.
- o Connection between Quality Control and data quality.
- o Relation between data quality and the value of data.
- o Organization-specific data integrity/data quality policies derived from a body of principles.
- o Cost-Benefit Relationships between security controls and data quality and integrity.
- o Organizational structures for assigning data integrity, quality assurance, and data quality functions.
- o A realistic internal audit role relative to data integrity and quality.
- o A reasonable external auditor role relative to data security and its subset data integrity.
- o The relation of people roles (ADP staff, user, internal auditor, quality assurance, quality control) to data integrity and quality.

Papers should be submitted by November 17, 1988 to:

National Institute of Standards and Technology
Computer Security Division
Attn: Zella Ruthberg
A-216 Technology
Gaithersburg, Maryland 20899

Approximately six papers will be selected for presentation and discussion. Selections will be made by December 7, 1988.

The strawman paper will be sent on request. For further background, people may also request a copy of the report of the 1987 WIPCIS workshop. The paper and report are available from Robin Bickel at the above address or 301-975-3359. For further information on the Workshop, contact Zella Ruthberg at 301-975-3361.

APPENDIX D

The Workshop Attendee List

APPENDIX D

ATTENDEES AT DATA INTEGRITY WORKSHOP

January 25-27, 1989

Key to last line of each attendee's data:
phone no. e-mail address fax no.

1. Marshall Abrams
   Principal Scientist
   The Mitre Corp.
   7525 Colshire Drive
   McLean, VA 22102
   703-883-6938 abrams@mitre.org 703-883-5687

2. Eugen Bacic
   Software Security Specialist
   Communications Security Establishment
   P.O. Box 9703, Terminal
   Ottawa, Canada K1G 3Z4
   613-991-7208

3. Robert Baldwin
   Tandem Computers
   19333 Vallco Parkway
   MS 3-15
   Cupertino, CA 95014
   408-725-7233

4. Joseph M. Beckman
   Computer System Analyst
   National Computer Security Center
   9800 Savage Rd.
   Fort Meade, MD 20755-6000
   301-859-4488 beckman@dockmaster.arpa

5. Thomas A. Berson, Pres.
   Anagram Laboratories
   P.O. Box 791
   Palo Alto, CA 94301
   415-324-0100 berson@kl.sri.com 415-324-0121

6. Deborah J. Bodeau
   Lead Engineer
   The Mitre Corp.
   Burlington Road
   Bedford, MA 01730
   617-271-8436 djb@mitre-bedford.arpa

7. David A.Bonyun
   Director of Computer Security
   AIT Corporation
   9 Auriga Drive
   Nepean, Ontario
   Canada K2E 7T9
   613-226-7800 dbonyun@ncs.dnd.ca 613-226-3066

8. Dennis K. Branstad
   Senior Computer Science Fellow
   NIST
   Technology A216
   Gaithersburg, MD 20899
   301-975-2913 dkb@ecf.icst.nbs.gov 310-948-1784

9. Nander Brown
   Agency Computer Security Program Mgr
   OIRM/SBA
   Rm 906A
   1441 L St., NW
   Washington, D.C. 20416
   202-653-6747

10. Rae K. Burns
    Principal Consultant
    Kanne Associates, Inc.
    219 Bragg Hill Road
    Ashburnham, MA 01430
    508-874-5291 burns@dockmaster.arpa

11. Peter G. Capek
    Research Staff
    IBM Research
    P.O. Box 218
    Yorktown Heights, N.Y. 10598
    914-945-1250 capek@ibm.com 914-945-2141

12. Thomas M. Chen, Mgr
    VS/OS Sec. Development Mgr
    Mail Stop 013-790
    Wang Laboratories, Inc.
    1 Industrial Ave.
    Lowell, MA 01851
    508-967-7684 tmchen@dockmaster.arpa

13. David Clark, Sr. Research Scientist
    M.I.T. Lab for Computer Systems
    545 Main St., Room 540
    Cambridge, MA 02139
    617-253-6003 ddc@lcs.mit.edu 617-258-8682

14. Robert H. Courtney, Jr. (Workshop Co-Chair)
President
RCI Inc.
Box 836
Port Ewen, N.Y. 12466
914-338-2525 rhcx@lanl.gov or courtney@ecf.icst.nbs.gov

15. Viiveke Fak, Assoc. Prof.
Dept. of Electrical Engrg
Linkoping U.
S-581 83 Linkoping
Sweden
46 13 281000 (o) v.fak@linnea.liu.se 46 13 139282

16. Ken Eggers
The Mitre Corp.
MITRE C3I Division
Washington C3I Operation
7525 Colshire Drive
McLean, VA 22102-3184
703-883-7080

17. Betty C. Hill
Information Security Officer
The World Bank Group
1818 H Street, N.W.
Washington, D. C. 20433
202-473-2790 202-4771572

18. T. J. Humphreys
British Telecom
St. Vincent House
1 Cutler St., Ipswich
Suffolk, UK
+44-473-224466 +44-473-214035

19. Assoc. Prof. Sushil Jajodia
Information Systems & Systems Engrg Dept
George Mason University
Fairfax, VA 22030-4444
703-764-6192 jajodia@gmuvax2.gmu.edu

20. Roy W. Jones (paper given, not present)
ICL Defence Systems
ICL (UK) Limited
Eskdale Road Winnersh
Wokingham Berkshire RG11 5TT
(0734)693131

21. Howard Johnson, Pres.
Info. Intelligence Sciences, Inc.
15694 East Chenango Ave.
Aurora, Colorado 80015
303-693-8291 howardj@csugreen (bitnet) or colostate.edu

22. Robert Jueneman, Dir. of Infosec. Technology
Computer Sciences Corp.
3160 Fairview Park Drive
Falls Church, VA 22042
703-876-1076 703-876-0878

23. Stephen Kane, Sr.
Product Mgr, Secure Systems
Wang Laboratories, Inc.
1 Industrial Ave.
Lowell, Mass. 01851
Mail Stop 013-A2A
508-967-0318 kane@dockmaster

24. Stuart Katzke, Chief
Computer Security Division
NIST
Bldg 225, Room A216
Gaithersburg, MD 20899
301-975-2929 katzke@st1.icst.nbs.gov

25. Stewart Kowalski
Researcher, Proj. for System Integrity & Information
Stockholm University
Dept of Computer & System Sciences
S-106 91 Stockholm
Sweden
46 8 759 4600 (o) stewart@suadb.se (internet)
46 8 162040 (u)uucp:mcvax!enea!suadb!stewart(univ.?)
46 8 159726 (h) 46 8 15 97 26 fax

26. Karl Krueger
Chief Information Security Officer
Room H2062
The World Bank Group
1818 H Street NW
Washington, D.C. 20433
202-473-2317 krueger@ibrdvm1 (bitnet) 202-4771572

27. Jan Kruys
Senior Consultant
NCR Systems Engineering B.V.
Zadelstede 1-10
3431JZ Nieuwegein
Netherlands
31 3402 76529 31 3402 39125

28. Milan S. Kuchta, Proj. Mgr
System Security Centre
Communication Security Establ.
P.O. Box 9703, Terminal
Ottawa, Canada K1G 3Z4
613-991-7331 kuchta@ncs.dnd.ca 991-7500

29. Stanley A. Kurzban, Sr. Instructor
IBM Corp.(798/TEC)
500 Columbus Ave.
Thornwood, N.Y. 10954
914-742-5638 kurzban@ecf.icst.nbs.gov

30. Carl Landwehr
Computer Scientist
Naval Research Lab
Code 5542
4555 Overlook Ave., S.W.
Bldg 16, Rm 111
Washington D.C. 20375-5000
202-767-3381 landwehr@nrl-css.arpa or @itd.nrl.navy.mil

31. E. Stewart Lee, Prof.
Computer Systems Research Inst.
University of Toronto, C.S.R.I., Rm 2002
10 King's College Rd.
Toronto, Ontario M5S 1A4
416-978-5035 stew@hub.utoronto.ca 416 978 4765

32. Theodore M. P. Lee
Trusted Information Sys., Inc.
P.O. Box 1718
Minnetonka, MN 55345-0718
or 15301 Willowood Drive
Minnetonka, MN 55345
612-934-5424 lee@tis.com

33. Steven B. Lipner, Mgr.
Secure Sys. Development
Digital Equipment Corp.
85 Swanson Road
BXB1-2/D04
Boxboro, MA 01719-13426
508-264-5090 lipner%ultra.dec@decwrl.dec.com 508-264-5100

34. Terry Mayfield, Assist. Dir.
Computer & Software Engrg Div.
Institute for Defense Analyses
5 Skyline Place, Suite 300
5111 Leesburg Pike
Falls Church, VA 22041
703-824-5524 mayfield@ida.org 703-845-2588

35. Kurt Meiser
Coopers & Lybrand
1251 Ave. of the Americas
New York, N.Y. 10020
212-536-1788 or 914-542-4280 MCI:351-6344

36. Dale W. Miller
Director of Consulting Services
Irongate, Inc.
7 Mt. Lassen Dr., Suite C-126
San Rafael, CA 94903
415-491-0910 415-479-0866

37. Roger L. Miller
Senior Programmer
IBM DB2 Design
IBM Santa Teresa Lab J67/B45
P.O. Box 49023
San Jose, CA 95161-9023
or 555 Bailey Ave.
San Jose, CA 95141
408-463-2371

38. William H. Murray, Fellow
Info. System Sec.
Ernst & Whinney
21 Locust Avenue
Suite 2D
or Stepping Stones
705 Weed Street
New Canaan, CT 06840
203-966-4769 whmurray@dockmaster.arpa

39. Lee Ohringer
Ass't Automated Info. Security Mgr
Department of Treasury
Bureau of Engraving & Printing
P. O. Box 5667
Washington, D.C. 20016
or 14th & C St. SW
Washington, D.C. 20228
202-447-0516

40. Carl A. Pabst
Principal Consultant in EDP
Touche Ross International
1000 Wilshire Boulevard
Los Angeles, CA 90017-2472
216-688-5201
or 17620 Rayen St.
Northridge, CA 91325
818-886-2506

41. Donn Parker
Sr. Mgmt. Sys. Consultant
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
415-859-2378 dparker@kl.sri.com 415-326-5512

42. Tom A. Parker
ICL DEFE Principal Consultant
ICL Defence Systems
Eskdale Road, Winnersh
Wokingham, Berkshire RG11 5TT
+44-249-693131 fax same phone, ask for fax extension

43. Tom Peltier (not present, sent paper)
General Motors Corp.
3044 General Motors Blvd
Rm 13-257
Detroit, Michigan 48202
313-556-1627

44. Paul Peters
Deputy Director
National Computer Security Center
9800 Savage Road
Fort Meade, MD 20755
or 8802 Churchfield Lane
Laurel, MD 20708
301-859-4372 ppeters@dockmaster

45. Sylvan Pinsky
Senior Scientist
National Computer Sec. Center
8331 Streamwood Drive
Baltimore, MD 21208
301-859-4485 pinsky@dockmaster.arpa

46. Sig Porter
Consultant in Security
614 Santa Alicia
Solana Beach, CA 92075
617-755-7814 sporter@dockmaster.arpa

47. Maria M. Pozzo
Member of Technical Staff
Aerospace Corp. M8\OSS
2350 E. El Segundo Blvd
El Segundo, CA 90245-4691
213-336-3096 pozzo@aerospace.arpa or pozzo@cs.ucla.edu
or 1013 Ocean Park Blvd. #8
Santa Monica, CA 90405
213-452-4904

48. David Rosenthal, Logician
Odyssey Research Associates
301A Harris B. Dates Drive
Ithaca, N. Y. 14850
607-277-2020 davidr@cu-arpa.cs.cornell.edu

49. Robert Rosenthal
Electrical Engineer
Computer Security Division
NIST
Technology A216
Gaithersburg, MD 20899
301-975-3603

50. Zella G. Ruthberg
Computer Scientist
NIST
Bldg 225, Rm A216
Gaithersburg, MD 20899
301-975-3361 ruthberg@ecf.icst.nbs.gov 301-948-1784

51. Damian Saccocio (observer)
Staff Officer
National Academy of Sciences
Computer Science & Technology Board
National Research Council
2101 Constitution Avenue, N.W., Room HA 560
Washington, D.C. 20418
202-334-2605 dsaccoci@nas(bitnet) 202-334-2791

52. Ravi Sandhu, Asst. Prof.
Dept of Computer & Info. Science
The Ohio State University
2036 Neil Avenue Mall
Columbus, OH 43210-1277
614-292-0394 sandhu@cis.ohio-state.edu

53. Marvin Schaefer, Vice Pres.
Trusted Information Sys. Inc.
3060 Washington Rd., Rte 97
P.O. Box 45
Glenwood, MD 21738
301-854-6889 marv@tis.com 301-854-6889

54. William R. Shockley, Dir.
Trusted Database Sys.
Gemini Computers, Inc.
P.O. Box 222417
Carmel, CA 93922
or 60 Garden Ct., Suite 110
Monterey, CA 93940
408-373-8500 408-373-5792

55. Gary Smith
Information Systems & Systems
Engineering Department
George Mason University
Fairfax, VA 22030-4444
703-764-6296(o) gwsmith@pentagon-opti.army.mil
703-803-6870(h)

56. Eva Sparr (submitted paper, not present)
RIKSREVISIONSVERKET
National Audit Bureau
Box 34105
S-10026 Stockholm
Sweden
46 8-7384193 46 8-56 04 25

57. Dennis Steinauer
Computer Scientist
NIST
A-216 Technology
Gaithersburg, MD 20899
310-975-3357 steinauer@ecf.icst.nbs.gov 301-948-1784

58. Stelio Thompson-Sittas
President/CEO
Expert Systems Software, Inc.
P.O. Box 15967
Long Beach, CA 90815
213-494-2573 or 213-499-3347(messages)

59. John M. Thurlow
I.S. Security & Control Coordinator
Exxon Corporation
P.O. Box 153
Florham Park, NJ 07932-0153
201-765-7095 fax 201-765-7612 (765-7021 for assistance)

60. Anne Todd
NIST
Bldg 225, Rm B363
Gaithersburg, MD 20899
301-975-3366 todd@ecf.icst.nbs.gov 301-948-1784

61. Douglas Varney
Kansas State University
417 Westview Drive
Manhattan, Kansas 66502
913-539-1510 or 913-776-5705 varney@ksuvax1.cis.ksu.edu

62. Grant M. Wagner
Chief Evaluator
National Computer Security Center/C12
9800 Savage Road
Fort Meade, MD 20755-6000
301-859-4458 gwagner@dockmaster.arpa

63. Dr. Willis Ware
Corporate Research Staff
The Rand Corp.
1700 Main St.
Santa Monica, CA 90406
213-393-0411 x6432 willis@rand-unix.arpa 213-393-4818
or rand.or

64. Stephen R. Welke
Research Staff Member
Institute for Defense Analysis
5 Skyline Place, Suite 300
5111 Leesburg Pike
Falls Church, VA 22041
703-845-3597 welke@ida.org 703-845-2588

65. Peter Wild, Mgr
Coopers & Lybrand
1251 Avenue of the Americas
New York, N.Y. 10020
212-903-3259

66. David R. Wilson
Principal
Ernst & Whinney, Inc.
2000 National City Center
Cleveland, OH 44114
216-861-5000 216-861-4966

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** (See instructions)	1. PUBLICATION OR REPORT NO. NIST/SP-500/168	2. Performing Organ. Report No.	3. Publication Date September 1989

4. TITLE AND SUBTITLE

Report of the Invitational Workshop on Data Integrity

5. AUTHOR(S)

Zella G. Ruthberg and William T. Polk

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
(formerly NATIONAL BUREAU OF STANDARDS)
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

Same as item #6

10. SUPPLEMENTARY NOTES

This document describes the results of the January 25-27, 1989 workshop on data integrity both via attendee reports and editor interpretation.

Library of Congress Catalog Card Number: 89-600756

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

This report contains the proceedings of the second invitational workshop on computer integrity issues and is the second response to the Clark/Wilson paper entitled "A Comparison of Military and Commercial Data Integrity Policy." The NIST Computer and Telecommunications Security Council established a Working Group on Data Integrity in late 1987 whose deliberations were the primary source for deciding on Data Integrity as the subject of the second workshop. The Planning Committee, composed primarily of Working Group members, outlined the scope of this workshop as discussions of 1). Integrity Framework Elements, 2). Implementation Requirements and Approaches, and 3). Implementation/Models in the light of the agreed upon integrity framework. The five discussion groups covered: Operating Systems and Systems, Telecommunications, System Services, Applications, and Implementations/Models. No consensus was reached on the definition of data integrity but consensus was reached on quality oriented 1). Policy and Objectives and 2). Mechanisms. Using the consensus, the five discussion groups came up with recommendations for future research.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

commercial computer systems; computer information system; constrained data item; data integrity; data quality; data value; integrity framework; integrity interface; integrity verification procedure; separation of duties; transformation procedure

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

377

15. Price

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents
Government Printing Office
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

☆ U.S. GOVERNMENT PRINTING OFFICE: 1989— 242 - 200/05071

NIST Technical Publications

Periodical

Journal of Research of the National Institute of Standards and Technology—Reports NIST research and development in those disciplines of the physical and engineering sciences in which the Institute is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Institute's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Institute's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NIST, NIST annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NIST under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published quarterly for NIST by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW., Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Institute on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NIST under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NIST administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NIST research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NIST publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NIST publications—FIPS and NISTIRs—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NIST pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NIST Interagency Reports (NISTIR)—A special series of interim or final reports on work performed by NIST for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, MD 20899