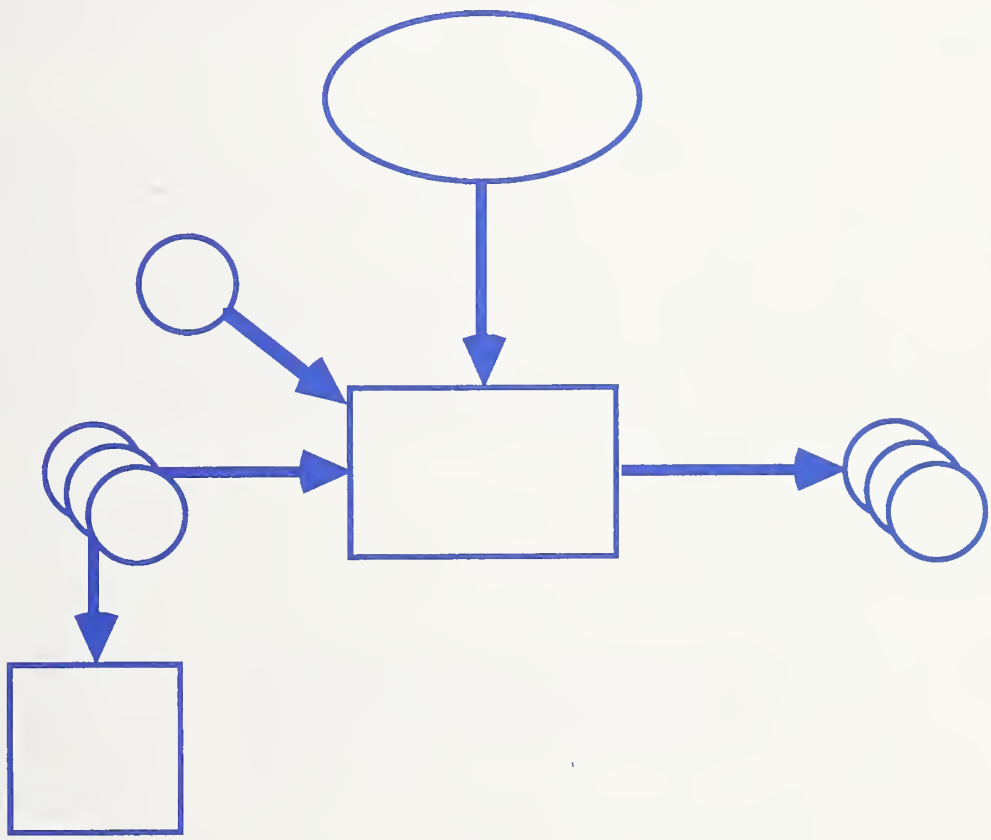
(formerly National Bureau of Standards)

Computer Science and Technology

NIST Special Publication 500-160

Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)

Dr. Stuart W. Katzke and Zella G. Ruthberg, Editors



Computer Science and Technology

NIST Special Publication 500-160

Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)

Dr. Stuart W. Katzke and Zella G. Ruthberg, Editors

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

Sponsored by:

ACM/Special Interest Group on Security, Audit and Control

IEEE/Computer Society, Technical Committee on Security and Privacy

National Computer Security Center
National Institute of Standards and Technology

January 1989

NOTE: As of 23 August 1988, the National Bureau of Standards (NBS) became the National Institute of Standards and Technology (NIST) when President Reagan signed into law the Omnibus Trade and Competitiveness Act.

U.S. DEPARTMENT OF COMMERCE
C. William Verity, Secretary

Ernest Ambler, Acting Undersecretary for Technology

National Institute of Standards and Technology
(formerly National Bureau of Standards)
Raymond G. Kammer, Acting Director

Library of Congress Catalog Card Number: 88-600606
National Institute of Standards and Technology Special Publication 500-160
Natl. Inst. Stand. Technol. Spec. Publ. 500-160, 195 pages (Jan. 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
INTEGRITY POLICY IN COMPUTER INFORMATION SYSTEMS (WIPCIS)

October 27-29, 1987
Bentley College
Waltham, Massachusetts

ABSTRACT

This is the Report of the Invitational Workshop on Integrity Policy in Computer Information Systems which was sponsored by the IEEE Computer Society's Technical Committee on Security and Privacy, the Special Interest Group on Security, Audit, and Control (SIGSAC) of the Association for Computing Machinery, the National Computer Security Center, and the Institute for Computer Sciences and Technology at the National Bureau of Standards. The workshop established a foundation for further progress in defining a model for information integrity. The workshop was held in response to the paper by David Clark of M.I.T. and David Wilson of Ernst and Whinney entitled "A Comparison of Military and Commercial Data Security Policy." The report's 10 sections contain an introduction, the composition of the organizing committee with a list of participants and a workshop agenda, a summary report by Donn Parker and Peter Neumann of SRI International, the reports of the five working groups, a response by Clark and Wilson, and a proposal by the National Bureau of Standards for continuing the effort to define an integrity policy. The appendices include a copy of the original Clark-Wilson paper, a summary of the Clark-Wilson rules, a number of position papers submitted in advance of the workshop, several papers submitted during and following the workshop, and a list of reference materials related to the integrity policy effort.

TABLE OF CONTENTS

Acknowledgments

The editors wish to thank Robin Bickel for her careful handling of all editorial changes in this machine readable document.

Note: Subsequent to the holding of this workshop, the name of the Institute for Computer Sciences and Technology at the National Bureau of Standards was changed to the National Computer Systems Laboratory at the National Institute of Standards and Technology. Because the text of this document was written when the old name was in force, all references to these names are in their old form.

REPORT OF THE INVITATIONAL WORKSHOP ON INTEGRITY POLICY IN COMPUTER INFORMATION SYSTEMS (WIPCIS)

October 27-29, 1987
Bentley College
Waltham, Massachusetts

1.0 INTRODUCTION

In April of 1987 a paper was presented to the IEEE Technical Committee on Security and Privacy by David D. Clark of M.I.T. and David R. Wilson of Ernst & Whinney. The paper suggested that previous work on data security had favored confidentiality policy in response to the needs of the defense establishment at the expense of integrity policy as required by the commercial and non-defense sectors. It proposed an integrity policy based upon separation of duties, well-formed transactions, and audit trail.

The paper created a great deal of interest in the security community and in order to further the approach, a committee was formed to convene an invitational workshop of about forty to fifty professionals to test and further the ideas presented in the paper. (The composition of the committee is included in the next section.)

The workshop was held October 27-29, 1987 at Bentley College in Waltham, Massachusetts and was sponsored by the IEEE Computer Society's Technical Committee on Security and Privacy, the Special Interest Group on Security, Audit, and Control (SIGSAC) of the Association for Computing Machinery, the National Computer Security Center, and the Institute for Computer Sciences and Technology at the National Bureau of Standards. (The participants are likewise listed in the next section.) It was opened in plenary session by David Callaghan, host and local arrangements chairman. After a number of presentations (see agenda in the next section) the workshop dissolved into five working groups plus a sixth group of floaters. The principal work of the workshop was done in the working groups; the floaters served to keep each working group aware of the direction and progress of the other groups. There was a progress report on the second morning and final reporting out on the third day.

The working groups were given both a general charge and one that was specific to each group. In general they were asked to test the Clark-Wilson Model on the basis of the requirement for such a model and how well the model met the requirement. They were also asked to test each rule of the model and the model as a whole, and to determine if the rule was properly stated and the model was complete. One group was asked to comment on the granularity and mandatory functions of the data objects and processes (transformation procedures), initial validation

procedures (IVPs), and constrained data items (CDIs). A second was asked to comment on end-user authentication in general and its effect on the C-W model. A third was asked to consider the issue of assurance, a fourth User Authentication, and the fifth Auditability.

This NBS Special Publication is the proceedings report for the WIPCIS and contains ten sections and several appendices. The sections consist of an introduction, the composition of the organizing committee with a list of participants and a workshop agenda, a summary report by Donn Parker and Peter Neumann of SRI International, the reports of the five working groups, a response by Clark and Wilson, and a proposal by the National Bureau of Standards for continuing the effort to define an integrity policy. The appendices include a copy of the original Clark-Wilson rules, a number of position papers submitted in advance of the workshop, several papers submitted during and following the workshop and a list of reference materials related to the integrity policy effort.

The National Bureau of Standards' proposals for continuing this effort (see Section 10) outlines a set of steps for continuing with the development of an integrity policy and a set of activities for implementing this policy in user applications. In addition to identifying additional issues, the Organizing Committee expects to solicit papers that will clarify and/or modify the Clark-Wilson view and form a starting point for discussion at the second WIPCIS workshop. The National Bureau of Standards will host and organize this second WIPCIS workshop at its facilities in Gaithersburg, MD.

REPORT OF THE INVITATIONAL WORKSHOP ON
INTEGRITY POLICY IN COMPUTER INFORMATION SYSTEMS (WIPCIS)

October 27-29, 1987
Bentley College
Waltham, Massachusetts

2.0 THE COMMITTEE

Sheila Brand
National Computer Security Center

David Callaghan
Bentley College

David D. Clark
M.I.T.

Stuart Katzke
National Bureau of Standards

Stanley A. Kurzban
IBM Corp.

Carl E. Landwehr
Naval Research Laboratory

Steven Lipner
Digital Equipment Corp.

William H. Murray
Ernst & Whinney

David R. Wilson
Ernst & Whinney

Mr. Paul Woodie
National Computer Security Center

2.1 PARTICIPANTS

D. Elliott Bell
Trusted Information Systems

Thomas Berson
Anagram Laboratories

Deborah Bodeau
Member of Technical Staff
The MITRE Corp.

Earl Boebert
Chief Scientist
Honeywell SCTC

Sheila Brand
Special Asst. to Director
National Comp. Security Center

David Brewer
Dept. Trade & Industry Commercial Computer Security Cell
Royal Signals and Radar Establishment

David Callaghan
Bentley College
Computer Information Sys. Dept

Peter Capek
IBM Research

Leslie Chalmers
The Bank of California

David Clark
Sr. Research Scientist
M.I.T Lab for C.S.

Dorothy Denning
Digital Equipment Corporation

John Fitzgerald
Ernst & Whinney

Cheryl Helsing
President
Cheryl W. Helsing, Inc.

Keith Howker
STC/ICL
ICL House

Robert Jueneman
Dir. of Infosec. Applications
Computer Sciences Corp.

Paul Karger
Digital Equipment Corp.

Stuart Katzke
Division Chief
National Bureau of Standards

Stan Kurzban
Senior Programmer
IBM

Theodore Lee
Trusted Information Sys. Inc.

Steven Lipner
Mgr, Secure Sys. Development
Digital Equipment Corp.

Kurt Meiser
Coopers and Lybrand

Dale Miller
Bank of America

Rolf Moulton
Mgr, Corp. Infor. Sys. Sec.
BP America

William Murray
Fellow, Info. System Security
Ernst and Whinney

Peter Neumann
SRI Int., Computer Science Lab

Donn Parker
Sr. Mgt. Sys. Consultant
SRI International

Paul Peters
National Comp. Security Center

Maria Pozzo
UCLA (also the Aerospace Corp.)
Computer Science Dept.

Marvin Schaefer
Trusted Informations Sys. Inc.

Jim Schweitzer
Xerox

William Shockley
Director, Trusted Database Sys
Gemini Computers, Inc.

Dennis Steinauer
Computer Scientist
National Bureau of Standards

Phil Terry
TSL Communications

Clark Weissman
UNISYS, Defense Systems

Peter Wild
Manager
Coopers and Lybrand

David Wilson
Ernst & Whinney

Simon Wiseman
Royal Signals & Radar Estb.

Paul Woodie
Network Integration and Test
National Comp. Security Center

Workshop on Integrity Policy in Computer Information Systems (WIPCIS)

TUESDAY, OCTOBER 27

7:15	Bus leaves Quality Inn for campus		
7:45	Buffet Breakfast		LaCava 325BC
8:30	Plenary Session:		Graduate Center Commons
	Welcome, Orientation	Dave Callaghan	
	The Program	Stan Kurzban	
	"The Paper"	David Clark	
	An Implementation	Earl Boebert	
10:30	Coffee Break		Graduate Center Commons
10:45	Plenary Session continues		Graduate Center Commons
	"A Contest for Integrity"	Bob Courtney	
	"The Workshop as a Beginning"	Stu Katzke	
11:45	Keynote Address		
	"The Work of this Workshop"	Bill Murray	
12:15	LUNCH		LaCava 325BC
1:00	Working Sessions		LaCava 305AB Grad Ctr. 161, 163, 164
5:30	Reception		LaCava 300
6:30	Dinner		LaCava 325BC
7:30	Bus returns to Quality Inn		

Workshop on Integrity Policy
in Computer Information Systems
(WIPCIS)

WEDNESDAY, OCTOBER 28

6:45	Bus leaves Quality Inn for campus	
7:15	Buffet Breakfast	LaCava 325BC
8:00	Interim Reports and Discussion Stan Kurzban	Graduate Center Commons
10:00	Coffee Break	Graduate Center Commons
10:15	Working Sessions	LaCava 300AB Grad Ctr. 161, 163, 164
12:00	Buffet Lunch	LaCava 325BC
1:00	Working Sessions	LaCava 300AB Grad Ctr. 161, 163, 164
6:00	Dinner	LaCava 305AB
7:15	Bus returns to Quality Inn	

Workshop on Integrity Policy
in Computer Information Systems
(WIPCIS)

THURSDAY, OCTOBER 29

6:45	Bus leaves Quality Inn for campus	
7:15	Buffet Breakfast	LaCava 325BC
8:00	Final Reports and Discussion Stan Kurzban	Graduate Center Commons
10:00	Coffee Break	Graduate Center Commons
11:30	Wrap-up	Graduate Center Commons
12:00	Bus leaves for Quality Inn for those needing to catch early flights	
12:00	Lunch	LaCava 325BC
1:00	Bus leaves for Quality Inn	

3.0 A Summary and Interpretion of the Invitational Workshop on Integrity Policy in Computer Information Systems 27, 28 and 29 October 1987 Waltham, Massachusetts

Donn B. Parker and Peter G. Neumann
SRI International, Menlo Park CA 94025-3493
Written 6 November 1987 for I-4 members
Revised 4 April 1988 for the WIPCIS report

This summary was originally written for the members of the International Information Integrity Institute (I-4) under the auspices of SRI International. It is reproduced with permission of the I-4 member organizations. It has been modified for inclusion in this report.

INTRODUCTION

The invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS), sponsored by the IEEE Computer Society, the Association for Computing Machinery (ACM), the National Computer Security Center (NCSC), and the National Bureau of Standards (NBS), was organized by Bill Murray of Ernst & Whinney, Stan Kurzban of IBM, and several others who had attended the presentation of the paper by David Clark and David Wilson, "A Comparison of Commercial and Military Computer Security Policies", at the 1987 IEEE Symposium on Security and Privacy. The workshop was attended by 38 computer security researchers, system designers, computer security managers from several large companies, and computer security consultants.

In their paper Clark and Wilson described a unified data integrity model that had created considerable interest in the computer security field. Some participants believed that the model could form the basis for integrity criteria in the sense that the Orange Book (TCSEC [85]) provides security criteria for trusted computer systems for the U.S. Department of Defense, as promulgated by the National Computer Security Center. Others sought to demythologize it.

The purpose of the invitational workshop, as stated in the initial information, was to respond to the ideas presented in the Clark-Wilson paper and to reach a consensus on a model or set of rules for data integrity in commercial data processing. Later, the term "commercial" was eliminated in order to permit application of the model to any organization engaged in secure use of computers where integrity is of concern as well as confidentiality.

The model and the workshop findings are of extraordinary interest. The concepts can be directly applied to day-to-day development of computer applications providing greater protection from fraud and abuse. In addition, several of the individual ideas could be applied to enhance information and computer security activities within a business organization. For example, implementation of the concept of separation of duties as a fundamental means of protection in financial and other transaction-oriented systems was described in the Workshop in complete and practical ways.

For the purposes of this document, we make a distinction among different types of policies. The two main types involve mandatory and discretionary controls, as is customary. However, two types of mandatory controls are considered here -- label-based mandatory (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, i.e., not requiring labels but also not permitting any user discretion over the use of the controls). For simplicity, the term "mandatory" when used alone implies the general case. (The Orange Book influenced the background thinking of many workshop attendees and was frequently referenced. For example, one discussion about mandatory versus discretionary policies reminded us of the natural language meaning of mandatory; the Orange Book definition is too narrow and usurps the natural language meaning. Workshop participants agreed to use "mandatory" in the more general sense, to imply a policy that is enforced by the system without user alterability. The rather awkward term "nondiscretionary" is also used elsewhere to connote label-based policies, e.g., Lipner [82].)

THE CLARK-WILSON INTEGRITY MODEL

The basis for the Clark-Wilson paper was that the Orange Book does not define adequate integrity policy for business applications. The Clark-Wilson integrity model attempts to define a (general-sense) mandatory integrity policy for such applications.

Although there had been considerable dispute on the relative roles of the Orange Book and Clark-Wilson before the workshop, the ensuing discussion during the workshop (described here) produced a view of general compatibility rather than antagonism. The key notion is that the Orange Book addresses the notion of security of an operating system or trusted subsystem that underlies a given application, while Clark-Wilson generally addresses the notion of integrity in the application itself. (David Wilson has suggested that Clark-Wilson could be applied to the integrity of the underlying operating system, e.g., using separation of duties for system configuration control, but there

is also a notion of system integrity in the Orange Book, so some overlap is evident.)

The primary goal of the Clark-Wilson data integrity model is to preserve the consistency and validity of structured data items both internal and external to an application. The Clark-Wilson model is described as a set of rules, of two types: C rules, for certification (although workshop participants agreed that was probably a misnomer), and E rules, for enforcement. The primary mechanisms of integrity are identified as well-formed transactions, separation (or segregation) of duties, and logging of transactions for audit purposes. Well-formed transactions are illustrated by consistently balanced double-entry bookkeeping operations that must be completed in an atomic fashion -- completely, or not at all.

The terminology used by Clark-Wilson is summarized as follows.

CDI = Constrained Data Item [Trusted data]
UDI = Unconstrained Data Item [Untrusted data]
TP = Transformation Procedure
[although it might have been Transaction Procedure or Process!]
IVP = Integrity Verification Procedure

The rules are summarized briefly as follows [where we have interpolated familiar concepts from the literature of software engineering to help in the explanations].

Clark-Wilson Enforcement Rules

E1. Users may operate on trusted data (CDIs) only through operations (TPs), never directly. ["Encapsulation" and "Abstract data types"]

E2. Users may perform operations only if explicitly authorized. ["Authorization"]

E3. User identities must be authenticated. ["Authentication"]

E4. Authorizations may be changed only by a security officer. ["Mandatory controls"]

Clark-Wilson Certification Rules

C1. Trusted data must be verified to be consistent representations of the real world. (IVP validates CDI state.) ["Data validation"]

C2. Programs implement operations as "well-formed" transactions. (TPs preserve valid state.) ["Consistent transformation"]

C3. The system must enforce separation of duties. ["Separation of privilege" and "allocation of least privilege", e.g., according to user roles]

C4. All operations (TPs) must be audited. ["Complete, nontamperable auditing"]

C5. Operations validate inputs, or reject them. (TPs validate UDIs.) ["Input validation" and "atomicity": Untrusted operation with untrusted data must be atomic-made valid or aborted.]

Separation of duties was described through the example of purchasing within a business, whereby the submission of a requisition, cutting of a purchase order, receipt of goods, and authorization of payment are each to be done by a separate person.

Separation of duties in a computer application requires (for example) that the programmer of the application system not be permitted to be a user of the program in a real production environment.

Assurance functions within the system are intended to reduce the cost of external assurance of correct functioning and to reduce the need to trust people unnecessarily by providing an application system in which engaging in fraud or causing errors would be difficult. The system can be preprogrammed and is highly predictable, whereas people cannot be preprogrammed and are not particularly predictable.

Logging for audit purposes is meant to assign responsibility and accountability for transactions after the fact. The logging process itself is a well-formed transaction and the log produces a genealogy of transactions.

In the Orange Book model security is based on doublets of subjects and objects, whereas the Clark-Wilson model integrity is based on a triplet of user, program, and data. Separation of duties is accomplished by strictly assigning users to programs and programs only to specified data bases. This is not

incompatible with the lattice model in the Orange Book, which permits different interpetations of subjects, but the triplet is not directly evident in the Orange Book. The integrity model requires that programs be unique items different from subjects and objects. One of the purposes of the integrity rule structure is to minimize the certification rules and maximize the enforcement rules, in that certification is generally much more difficult to achieve than enforcement.

The application of this model both internally within the system and externally among the users of the system presents an important dichotomy. The concept of an external data item was identified as valuable in a computer system for which there is an external equivalent. A difficulty with the model involves questions about the meaning of a partially completed sequence of transactions and the capacity of the model to deal with this.

DEMONSTRATIONS OF IMPLEMENTABILITY

A new computer system, LOCK, is being developed by Honeywell (distinct from the Honeywell-Bull computer manufacturer). (The LOCK effort is sponsored by the National Computer Security Center, and its results will be available to all --including other interested vendors.) This is the first attempt to develop a computer system that could be certified beyond the A1 level of the Orange Book. Earl Boebert of Honeywell showed how readily LOCK can implement the Clark-Wilson integrity model, as well as meet all requirements of the Orange Book and beyond. This discussion demonstrated that integrity and confidentiality are closely related, and can be compatibly achieved in the same system. However, integrity is provided as an extension of the basic kernel that enforces confidentiality, and uses LOCK's strong typing mechanism to ensure what might be called type-based mandatory integrity. (This is a less restrictive notion of mandatory integrity than the label-based mandatory integrity proposed by Biba [77]. We note here that a Biba-like two-level label-based mandatory integrity policy is found in Clark-Wilson, separating the trusted CDIs from the potentially untrusted UDIs, but requiring a Trusted Process in the Orange Book sense [namely a Transformation Procedure in Clark-Wilson -- conveniently acronymed TP!] to convert UDIs to CDIs. Sceptics of the significance of label-based mandatory integrity and security, even in unclassified applications, should see the important paper by Lipner [82] cited at the end of the bibliography.)

LOCK easily accommodates the closure concept of encapsulating a program and all the data that it uses and processes. For example, the Clark-Wilson model proposes that a program completely encapsulate its own objects so that they cannot be directly accessed by any other programs. By establishing types and domains properly, LOCK enforces this

desired encapsulation through the type mechanism and other system controls. The Honeywell LOCK approach uses hardware specially designed for security intended to go beyond the Orange Book A1 rating. (See the cited contributions by Earl Boebert.)

Other position papers on the implementability of Clark-Wilson on top of TCB-oriented systems were presented by Ted Lee and by Bill Shockley. Boebert, Lee, Shockley, and other participants concluded that given suitable mandatory security TCBs, the implementation of Clark-Wilson could be achieved with relative ease and could operate efficiently.

It is extremely interesting to note that all three of these have no trouble in accommodating the Clark-Wilson triplets, i.e., they go beyond what is required by the Orange Book.

Security is a reactive process that continually escalates. That is, a threat not met is addressed with a new security refinement, which leads to a new threat and a new refinement, and so on. Consequently, better understanding of the potential threats is necessary to improve security but cannot guarantee it.

WORKSHOP GROUP TOPICS ASSIGNED

The workshop was organized in five groups to address the Clark-Wilson rules:

(1) Reality (agreement of the internal model with its external environment) -- Rules C1, C3 and C5.
(2) End-user authentication -- Rule E3.
(3) Assurance -- Rules E1, E2, C2 and C5.
(4) Granularity -- Rules E2, E4, and C3.
(5) Logging for auditability -- Rule C4.

Each group initially addressed the following Questions. These questions have been heavily paraphrased because of the ambiguities in the original statements. [For the benefit of the reader, we also anticipate the answers generally agreed upon by the participants at the end of the workshop.]

* Is the Clark-Wilson model an appropriate one? [It is a useful first approximation, but needs much more work.]

* Is the model consistent within itself (and perhaps complete)? [No, it is neither completely consistent nor complete.]
* Are the individual rules appropriate, and properly stated? [Many of the rules are indeed appropriate, and do reflect a meaningful sense of integrity.]

* Does the model satisfy the requirement for integrity? [Partially.]

* What enforcement mechanisms are implied such as closed systems, and which rules are mandatory -- and in what sense? [Effective implementation requires some sort of underlying trusted computing base; furthermore, the implementation of the integrity rules requires a trusted environment. The integrity rules are indeed mandatory in the general sense.]

REALITY GROUP

The reality group discused how a computer system represents reality (the real world in which it functions). The importance for achieving this is heightened by the strategic use of computers in electronic data exchange among enterprises, where sources of data and output are widely disparate. Diagramming a large banking system network is no longer a reasonable task, because of the continual and undocumented extensions of the network.

This section seems disproportionately long compared with the others that follow. However, much of the background necessary for understanding of the other groups is included here, so we have not attempted to equalize the apparent coverage given each group.

Security Domains. Semantic and syntactic issues of data were discussed. Layers of abstraction are important to consider because definitions of losses and control objectives may have the same names but have different meanings depending on the abstraction layer. A concentric set of rings constituting domains within a business was proposed, whereby the management and participants at each layer are accountable for the integrity of the data in that layer. To define the perimeter of a domain, the concept of reversibility of conversion to loss was suggested: for every transaction within the domain, a corresponding transaction must occur to prevent loss; outside that domain the loss would be irreversible.

In developing the concept of domains, the group addressed the question of how to define the immediate domain of the system so that an application could be tested for integrity against the reality of the external world that it represents. The domain is defined by:

(1) The structure of the organization engaged in the use of, and affected by, the application.

(2) The potential threats the security model is to counter.

(3) A scope sufficient to encompass all effects of, and methods for, reversing a loss or impending loss.

(4) All motives, functions, and capabilities for separation of duties as required by the security model.

(5) All other concerns that have an effect on or are affected by the application.

After the domain of reality has been determined, the control objectives can be identified relative to the potential threats and risks to complete the model.

Separation of Duties. Once this framework was established, the group addressed separation of duties and its meaning within this concentric domain structure. In accounting, separation of duties is illustrated by one worker having access to assets, a different worker having access to accounting records of those assets, and a third worker being responsible for authorization. A definition of separation for our purposes was the assignment of functions to different people that would achieve conflicts of motives and independent confirmation of results of the transaction activity. Separation is adequate when no harmful action can be instigated by a person acting unilaterally. Every verification process must involve more than one person, and the separation of duties must produce the greatest possible conflict in the motives and abilities of the persons involved. The test of adequacy in the latter case is that two people who could collude would have the least likely motives and abilities to do so.

An attendee suggested, however, that system attackers seek well-defined domains and are happy with fixed security perimeters that identify areas to avoid in their attacks.

The subject of measuring the strength of separation of duties was introduced. The weakest form of separation would be dual control, for example, identical data entry by two persons. (An even weaker use -- or misuse -- would be the case in which the same person or process was able to gain control over two separated duties, perhaps through indirect paths or by exercising a system flaw.) The strongest form of separation would be the assignment of tasks to a larger number of individuals with the greatest differences in motives and abilities.

The group considered means of defining the strength of separation of duties and developed the following examples of how, stronger separation might be accomplished:

(1) Break down the duties into smaller segments assigned to more people.

(2) Require that certain duties be assigned dynamically and randomly to different sets of people so that in any complete transaction no information worker would know which sets of people

are performing it.

(3) Periodically rotate workers' assignments so that workers would never know when a coworker would be replaced with someone else.

(4) Reduce the total value of exposed assets to any one worker.

Research topics recommended by the group included developing integrity labels for cross-domain movement of CDIs; establishing a metric of strength for separation of duties; examining the possibility of putting more substance into invariant enforcement (E) rules rather than application-oriented certification (C) rules; and expanding the scope of the model to cover the security purpose of availability as well as integrity and confidentiality. On this last point, a group member commented that availability is in a trade-off relationship with integrity and confidentiality.

Others supported the extension of the model to include all three purposes (or the incorporation of the model into a more general model that contains all three), citing the complaints that the Orange Book addresses only a subset of the security purposes. Therefore, if the Clark-Wilson model is to be successful, it must overcome this shortcoming by addressing all three purposes.

Well-formed transactions. The relevance of the well-formed transaction to reality of the world it represents was considered. One of the properties of a well-formed transaction is that integrity verification is minimal in terms of balancing the results of the functions it performs. Such an audit requires that an audit be performed in real-time systems. If a snapshot is taken at arbitrary times, this can be extremely complex -- because the system is likely to be out of balance in a global sense at any point in time. A recommended audit technique is to take a snapshot at Time T1; all balancing would be accomplished except for suspense items resulting from transactions incomplete at T1 and from transactions that require others before a balanced state is achieved. A snapshot would then be taken at the next time increment, T2. Suspense items from T1 would be cleared if balanced from T2 information. At Time T3, additional suspense items would be reconciled for T1 and T2. Ultimately, all suspense items for T1 would be accounted for and complete balancing could occur. (Note that theoretical situations exist in which balancing will never occur, but these generally represent poorly designed systems. Note also that in certain applications, much more reliable strategies can be developed, using knowledge about the application.)

Threat Analysis. A presentation on a computer abuse loss event model and taxonomy of computer methods stimulated a discussion of threat analysis of the Clark-Wilson model. The purpose would be to determine whether the model covered all known threats to flaws in the model that would necessitate changing the model or extending its scope.

A participant asked whether the well-formed transaction concept and separation of duties are sufficient to assure the integrity of an application. A threat analysis might indicate that additional mechanisms are required to achieve this. For example, a potential threat might be the theft of the entire system from the domain, an irreversible loss. An additional mechanism on the physical computer system might be required to signal the operating system and controls in it that the system has been removed from its authorized site; when power was reestablished, the system would automatically go into a protective mode. This brought up the idea of instrumenting the domain of reality of a system to assure the integrity of that domain during system operation.

The group concluded that the most it could achieve in this workshop would be to recommend that a methodology be developed for testing the model against all known experience of loss, to prove its effectiveness.

Other Discussions. The possibility that CDIs and UDIs might cross domain boundaries was considered. One possibility was the need for tokens or labels for each UDI/CDI identifying its domain source and domain location.

Also considered was the possibility of requiring a specialized form of IVP (integrity verification procedure) that would perform a reality check of a CDI relative to the world it represents in the domain. This is the practice of confirmations in accounting audits, and the IVP could be similar. Auditors send confirmation slips to ensure that the party ultimately benefitting from or affected by a transaction has in fact been satisfied. The IVP would be a means of checking cross-domain integrity as long as the clocks are synchronized from one domain to another and the logs are available.

A question that arose in considering the reports of other workshop groups concerned the process in the model of transforming from one valid state to a new valid state. For any number of reasons, a valid state could become invalid. Therefore, it may be necessary at any time during the application execution to transform an invalid state into a valid state. The group concluded that correcting TPs may be required. Two possibilities arise: the application system is in balance but still an error has occurred, and the balancing is not valid after

an error has occurred. The question is whether the processes are reversible.

USER AUTHENTICATION GROUP

The user authentication process group developed the following set of rules for user authentication:

o Every user ID must be unique within the reality domain for the retention period of the audit log.

o Every user must have an ID. Each can have multiple IDs but only one per reality domain.

o The confidentiality and integrity of the authentication server and communications must be maintained.

o The authentication server must be considered as a TP with an encapsulated database.

o Enrollment must add to the base each name, ID, authenticator, and enrolling authority.

o IDs must be reconciled periodically against reality.

The problem of name changes must also be considered, as when a person divorces or marries arose.

The group also concluded that a separation of duties is required, (e.g., user and enrollment administrator). Areas identified for further research were privacy implications and estimating user security profiles.

ASSURANCE GROUP

"Assurance" is defined as the ability to reason about a system and predict its behavior. The group developed its recommendations from good engineering practices, as follows:

(1) Postulate the "properties" (e.g., Clark-Wilson rules). The properties must be invariant across systems; the Clark-Wilson model is an example of an invariant set of properties. The model would be an application. Fundamental errors must be avoided, and this can be aided by defining the security perimeter within which controls are affected and outside which there the implementing authority has no control. Part of this step is performing a stress analysis or testing of the postulates, and from this the specifications are derived .

(2) Factor the specifications into layers. In the Clark-

Wilson model, one layer would be the E Rules, which must be underlying, invariant, complete, and enclosed. A higher layer would enforce the C Rules, which would be adapted to the application system.

(3) Decompose the TP, IVP, CDI, and UDI and diagram these entities.

(4) Provide unit assurance of these decomposed parts.

(5) Recompose the TP, IVP, CDI, and UDI and perform subsystem assurance.

(6) Iterate on each of these steps to recompose the entire application system.

The group had some serious doubts about these steps. For example, in steps (2) and (3) the structures may be changing rapidly; a dynamic situation makes these steps difficult. A severe doubt about success is with step (5): Is a group of well-formed transactions in itself a well-formed transaction? Many experts at the workshop agreed that this is the most difficult step.

The most severe doubt concerned whether the notion of a security perimeter could be achieved at all, and whether the inductive hypothesis given in the Clark-Wilson paper was sound for any but toy systems. The spokesman for this group concluded that there are "lots of toy systems that seem to look secure." Full-scale demonstrations of successfully designed secure systems are needed. Someone from the audience suggested that the fact that 747s can fly is proof that a complex system can be successful even if it is not formally verified! But the 747 went through assurance steps that are very similar in concept to formal assurances, and if those steps had not been taken, the 747 probably wouldn't fly! But the spokesman remarked that anything as sloppily implemented as OS/360 would not fly. The response to that was something like OS/360's don't have to fly, only operate a computer on the ground, i.e., the assurance requirements are not that high.

Another comment was that the availability function of restarting the system should be in the discipline of system reliability rather than in security. The meaning of the term "reconstructed" in the C4 rule was clarified. It should be the reconstruction of the audit trail to determine whether a loss has occurred or to answer other audit questions and not to be considered the reconstruction of the application after a failure.

GRANULARITY GROUP

The group on granularity concluded that the model rules are reserved for management not for security officers. The security officer is a special surrogate of management. The group also determined that TPs are, in fact, CDIs and developed the following specifications:

o TPs must be small enough to preserve minimum granting of authority.

o TPs must be small enough to categorize in terms of origination, procedure, assets, approval, maintenance, data, and records.

o TPs must be reasonably sized in terms of other units of logic and programs.

o Limited, well-formed, and single-function are attributes of TPs. (There was disagreement on this point.) TPs will not work without an underlying trusted computer base (e.g., superzapping is a successful attack method if an organization does not have a trusted computer base).

o TPs and CDIs must have owners for granting and permit authority. Ownership transference and repudiation or withdrawal of protocols must be possible. Assigning ownership incorrectly is better than not assigning it at all. (?)

The group on granularity concluded that it did not know how to address the availability purpose of security along with integrity and confidentiality (this purpose was called continuity by the group).

The spokesman had one last admonition that if words are chosen for special meaning, they should still retain their English dictionary meanings. As an example, "sensitive but unclassified" is a classification that should be abandoned.

AUDITABILITY GROUP

The group on auditability reported that the four related purposes of logging and audit trails are to facilitate:

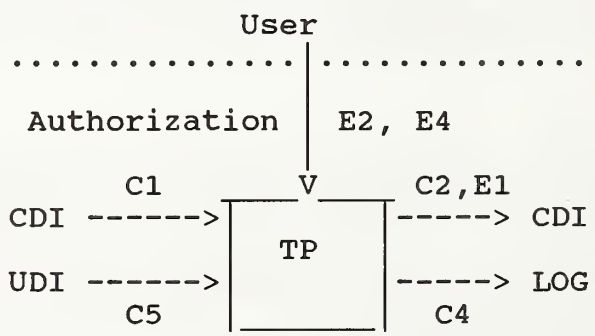
(1) Recovery following an abnormal system or application state
(2) Post-hoc auditing
(3) Real-time system monitoring
    (a) For threat monitoring
    (b) For performance monitoring
(4) Accounting and billing.

The group noted that the logging mechanisms and databases must themselves be nontamperable (e.g., nonbypassable and nonalterable).

Any application of the Clark-Wilson model must be based on some underlying trust, because the auditability requirements of the model require user process identification, defined interfaces, event logging, and protection of the logging mechanism and recording media. Logging must take place according to the object information being recorded. In the language of abstract data types and object-oriented computing, each data or object type has its own needs for logging. Changes were recommended in the wording of Rules C2, C4, and C5 that are described in the study group report (Peter Wild, Chairman). That document suggests that Rule E3 must be provided by the TCB and that E1, E4, and C3 were covered by other rules, e.g., as implementation instances.

One of the most significant changes in rules was the suggestion to replace C1 with the statement, "for every CDI there must be a set of IVPs that ensures that the CDI is in a valid state at some defined point in time." This is essentially an audit procedure, but it is not necessarily restricted to an auditor's use. Execution of an IVP is a loggable event. The group noted the redundancy of three rules -- Rules E1 (which is subsumed under E2), E4 (which is an implementation instance of E2), and C3 (which is an implementation instance of E2). These three rules could be deleted and subcased as indicated, although they may be useful as special cases for clarity.

A considerably simplified figure when compared with the figure in the original Clark-Wilson paper (p. 192) for the System Integrity Rules was recommended, based on the observation that an IVP is really a special case of TP. The resulting suggested diagram looked like this:



Authentication E3
Separation of duties C3:
(a) in design; (b) dynamically

Note that in order to provide assurance for the integrity of the implementation of the Clark-Wilson layer itself, some sort of Trusted Computing Base is required, to prevent compromise of the implementation. For example, there must be noncompromisable mechanisms for separation of duties (C3) and authentication (E3), upon which the Clark-Wilson implementation must depend. For auditing to be noncompromisable, the integrity layer must itself be protected from its users. This can be done most effectively by an underlying TCB, e.g., an Orange Book TCB for security, and otherwise by the Clark-Wilson Integrity layer itself.

It is important to keep in mind the distinction between a policy item (e.g., separation of duties) and the mechanisms that enforce it. In the case of separation of duties, design decisions must be made on when and what to enforce in the general policy; mechanisms must also exist to enforce that policy (presumably in the underlying TCB, to prevent compromise).

In general the logging group believed that the results of the entire workshop would have been more accessible to the workshop participants if the model had been couched explicitly in the framework of abstract data types and commonly used software engineering concepts. In that case, the Clark-Wilson rules would have been much clearer, as would the dependence upon some sort of underlying security TCB for implementation of the above-depicted integrity TCB, together with the relationships among mandatory and discretionary policy issues. Such a recasting was strongly recommended by the auditability group, and is suggested in our annotated list of the Clark-Wilson Enforcement Rules and Certification Rules given above, which hints at the notions of encapsulation and information hiding, strong typing, and hierarchical abstraction.

WORKSHOP WRAP-UP SESSION

The wrap-up session was organized so that only negative comments were permitted. This gave the advantage in conclusions to those with the most negative positions on further progress on the Clark-Wilson integrity model.

A tentative conclusion of the meeting was attempted: "The model is a coherent and useful foundation that has now been laid for further progress." This tentative conclusion was destroyed, however, by the negative comments. The general belief was that the results of the workshop were only a bare beginning toward a coherent and useful foundation. The participants achieved not much more than agreement on a common vocabulary, and did not reach closure on the completeness of the model. No significant examples of the concepts developed have been produced to demonstrate even the viability of this type of model. One participant pointed out that the current scope of applicability

of the model is unstable, considering that it addresses only one of the three security purposes. This was reinforced by comments from others. Someone mentioned that the Clark-Wilson model has opened Pandora's box; for example, we do not even know what a CDI is yet. Another comment was that the contents of Robert Courtney's letter (part of the WIPCIS report) are quite compelling.

OUR OWN CONCLUSIONS

This workshop was extremely valuable for attendees and the information security community. It provided a dialog and created greater understanding between the "scientists" and the "practitioners" in our field. Just as the Clark-Wilson Model is an important attempt to combine concepts, the workshop combined ideas about information systems from the business sector with concepts from the military and general security research sectors.

The participants agreed that the Clark-Wilson model requires considerable refinement, extension to the other purposes of security at the application layer, and more rigorous and complete definitions of items in the model. In addition, it can be adequately secure in a serious sense only if applications using the model concepts are executed in a protected domain, and optimally on top of an adequate trusted computing base. However, in the less-than-perfect environment of current business systems, invoking only some of the model concepts in new business applications can materially increase integrity -- as long as the underlying operating system cannot be compromised. (There is strong sentiment in the research community that it is dangerous to have a secure application running in a system that can be easily overcome by system programmers or intruders by accessing the underlying hardware and operating system layer, e.g., by superzapping, by becoming superuser, or by exploiting system flaws.)

In the final analysis, there is much progress being made with respect to the Clark-Wilson Model and various other security/integrity models under development or previously published. The NCSC Criteria Books (currently the Orange Book and the Red Book -- TCSEC-TNI [87] -- together with a Trusted Database Interpretation in preparation), the ECMA Framework, UK/DTI, and the ISO OSI Reference Model reflect steps in the right direction (more or less) --although there are still many problems. Some of these concepts are practical and usable today for increased security and integrity -- even though imperfectly. Others will not be effectively utilized until they appear in systems that are more secure, e.g., rated at least at the Orange Book B level. However, it will be essential to integrate into a common system framework all of the vital security-related

requirements -- confidentiality, integrity, availability, performance, reliability, some notion of correctness, etc.

The opinions represented here are those of the authors, and do not reflect any official positions whatever. Any misrepresentations of what actually transpired at WIPCIS or factual errors are ours.

SOME PRACTICAL IDEAS

(*** This section was written for I-4 members, but is included here for possible general interest. ***)

Some practical ideas are suggested below that may be amplified through reading of the WIPCIS report and backup material listed in the bibliography.

o The Clark-Wilson model along with suggested changes can be used as the basis for writing effective guidelines or improving existing guidelines for business-application development groups, to assure more-secure applications with better controls.

o Familiarity with the model and its concepts can provide computer security personnel with a guide and checklist when they participate in requirements, specifications, and design efforts for new or revised applications. For example, facilitating separation of duties externally can be supported by appropriate data entry design and controls derived from the model. For example, returning the full-text meaning of parameter values chosen can reduce error: Input by the data-entry operator of Customer Number "087643-1" returns

> "XYC Corp.
> Receiving Dept.
> Attn: Joe Danks
> P.O. Box 433
> Anytown, PA 00484"

to the supervisor who is making decisions about delinquent accounts.

o Guidelines or company standards are needed to establish separation of duties among information system users. This could be especially important for departments with many microcomputer users because they tend to use informal, careless practices, assuming that the computers somehow solve problems with minimal human controls. A spectrum of separation practices is suggested, from weak to strong:

Weak

- o Work sample reviews
- o Dual controls
- o Segregation of workers for each sequential transaction function
- o Frequent rotation of segregated workers
- o Different but equivalent data integrity checks
- o Dynamic transaction function assignments
- o Random control of workers rotation

Strong

o Reviews of current production systems comparing controls with the Clark-Wilson Model are an excellent way to identify security shortcomings. This was done by Bill Murray for the IBM AAS application (see ``Data Integrity in a Business Data Processing System'', W.H. Murray, 24 Oct 87).

o Awareness of and movement toward adoption of different security criteria in different market segments will cause great problems for computer manufacturers and could result in export exclusion internationally. Such impositions could cause severe problems for customers as well as security criteria incompatibilities among systems. Vendors and users alike must start exerting their influence to cause convergence rather than the current divergence.

BIBLIOGRAPHY

WIPCIS Documents:

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

2. O. Tami Saydjari, Joseph M. Beckman, Jeffrey R. Lesman, Locking Computers Securely, NCSC 1986.

3. W.E. Boebert, Practical Alternative to Hierarchical Integrity Policies, Proceedings of the Eighth National Computer Security Conference, 30 Sept 85.

4. W.E. Boebert, LOCK Implementation of the Clark-Wilson Rules, WIPCIS, 27 Oct 87. Vugraphs Only.

5. T.M.P. Lee, Using Mandatory Integrity to Enforce "Commercial" Security, (Draft) Trusted Information Systems, Inc., 26 Oct 1987.

6. W.R. Shockley, Implementing the Clark/Wilson Integrity Policy using Current Technology, Gemini Computers Inc., prepared for WIPCIS, Oct 87.

7. W.H. Murray, Data Integrity in a Business Data Processing System, 24 Oct 87.

8. W.H. Murray, Position Paper for Working Group on Granularity, WIPCIS, 27 Oct 87.

9. W.E. Boebert, Outline for the Verification Working Group, WIPCIS, 27 Oct 87.

10. David R. Wilson, Agreement with the External Environment, WIPCIS, 27 Oct 87.

11. Peter Wild, Availability Issues, Final Report, WIPCIS, 29 Oct 87.

12. W.H. Murray, Working Group on Granularity and Functions, Final Report, WIPCIS, 29 Oct 87.

13. Donn B. Parker, Proposal to Extend the Scope of the Clark-Wilson Model to Include Availability, WIPCIS, 29 Oct 87.

14. Donn B. Parker, Clark and Wilson Commercial Integrity Model Slides, SRI International.

15. WIPCIS List of Invitees to IEEE/ACM Invitational Workshop on Trusted Commercial Systems Criteria.

16. WIPCIS Participant Listing, WIPCIS, 29 Oct 87.

17. European Computer Manufacturers Association, Framework for Secure Open Systems, Second Draft, Aug 87.

Other Relevant Documents:

TCSEC [85] DoD Trusted Computer System Evaluation Criteria, Department of Defense, National Computer Security Center, DOD 5200.28-STD, Dec 1985. (The Orange Book)

TCSEC-TNI [87] Trusted Network Interpretation [of the TCSEC], National Computer Security Center, NCSC-TG-005 Version-1, 31 July 1987. (The Red Book)

Biba [77] K.J. Biba, Integrity Considerations for Secure Computer Systems, USAF Electronic Systems Division, Bedford MA, ESD-TR-76-372, April 1977.

Lipner [82] Steven B. Lipner, Non-Discretionary Controls for Commercial Applications, Proc. 1982 IEEE Symposium on Security and Privacy, pp. 2-10, 26-28 April 1982.

4.0 REPORT OF THE WIPCIS WORKING GROUP ON ASSURANCE

by Earl Boebert

Group Members: Deborah Bodeau, David Clark, Robert Jueneman, Theodore Lee, Steven Lipner, Dale Miller, Maria Pozzo, William Shockley

1. Introduction

The working group defined assurance as the ability to reason about a particular system or to predict its behavior. We took the position that assurance was always a matter of degree, and attempted to draw conclusions that were independent of specific assurance technologies, such as testing or formal methods.

We observed that in general, the preconditions for methodical testing, informal analysis, and formal verification of a computer system or subsystem are the same: namely, that a series of increasingly detailed design descriptions exist. These descriptions are related at one end to objective and meaningful requirements and at the other to the implementation. The assurance process consists of generating and analyzing evidence that each iteration of increased detail is sound.

We noted that assurance is achieved by good engineering practice: orderly development, in steps, with the results checked at each step. We noted that the steps in this process are traditionally described as if they happen in sequence; in actual fact they almost always happen iteratively.

The major steps in the development process, and the assurance steps that go along with them, form the framework for our report. We begin by sketching the development and assurance process, and then present, for each step, the doubts raised in our minds by Clark and Wilson's definitions of integrity. For each doubt we then issue a challenge in the form of a proposed research activity that we feel will address the area in question.

2. Development and Assurance

2.1 Demonstration of Invariant Properties

The first step in the process is to postulate a set of properties, or invariants, that the system must exhibit: things that happen with suitably small (approximating "never") or large (approximating "always") probability. The desirability of these properties is seldom in dispute when they are discussed in general terms (e.g., "safety," "security," or "integrity"). An example is the midair failure of a commercial airliner's structure (an instance of "safety"). The specified probability

of this occurrence is so small that it should "never" happen within the service life of a given aircraft.

Once a property is selected and a definition established (e.g., the actual allowed probability of structural failure) then the system to be constructed is modelled in some suitable mathematical notation, and a mathematical argument is presented that the model exhibits the properties. Continuing our commercial aircraft analogy, this first step is the stress analysis of the structure, which is based on an abstract representation of the aircraft. The step in no sense guarantees a safe airplane because people do not fly in abstract representations. Weaknesses can be introduced in later development steps through errors in detailed design, poor materials, or shoddy workmanship. The level of assurance at this point is that the aircraft is not inherently unsafe in a way that no amount of skill in detail design or construction can overcome.

The output of this step is a specification for the entity to be constructed which is consistent in the sense that it does not contradict itself and is relevant in the sense that it describes a useful and feasible system.

The importance of this first step should not be underestimated. Without this step, the project runs the risk of building an inappropriate system that is entirely correct (in the sense that the implementation meets the specifications), but does not perform any useful function. The specification may be thought of as defining a family of systems, that is, those that correctly implement the specification. The goal of the first assurance step is to show that any member of this family will have the desired properties.

In the case of the properties proposed by Clark and Wilson, the entities of interest are computer information systems which perform specific tasks with integrity. In the remainder of our report we will call these entities "applications subsystems," to distinguish them from other parts of the computer information system such as the hardware. We note that this definition is inherently imprecise, in that the entities discussed by Clark and Wilson involve elements of both the automated system and the organizational context in which it resides. This imprecision, as we shall describe later, was a source of difficulty.

We also note that the term, as we use it, encompasses a vertical "slice" of the system from the user interface to the lowest levels, and does not refer to just a program "sitting on top of" a general-purpose computer system. This latter point is important when considering the general topic of assurance: The degree of assurance that can be associated with the applications subsystem depends upon the degree of assurance that can be associated with the underlying facilities. An applications

subsystem consisting of strong programs using the facilities of a weak operating system inherits the weakness of the operating system and not the strength of the programs.

We also restricted our consideration of invariant properties to those which fall under the general term "integrity," and specifically integrity as defined by Clark and Wilson.

2.2 Factoring and Allocation of Requirements

The specification developed in the above step incorporates basic decisions about the functional characteristics of the applications subsystem and the integrity properties it is to exhibit. That specification is then "factored" into levels and major requirements implied by the specification are assigned to the various levels. Typical levels encountered in computer information systems (from the bottom up), are the hardware, the operating system nucleus, general operating systems services, protected subsystems, and user programs.

We noted that Clark and Wilson's framework strongly implies a general allocation of the "E-rules" to an applications-independent enforcement mechanism and the "C-rules" to the applications subsystem itself. We then considered the nature of the enforcement mechanism. We noted that elementary support for protected subsystems, such as a simple two-state processor and segmented memory management, is not sufficient for an enforcement mechanism in all cases. These features do not protect against "active" or "technical" attacks by programmers, because while they provide assurance that the enforcement mechanism cannot be tampered with, they provide no assurance that the enforcement mechanism cannot be bypassed.

We examined, to the degree that time allowed, the integrity mechanisms of a range of Trusted Computing Bases (TCBs) and discovered several whose built-in policies could, enforce the "E-rules" of Clark and Wilson. The mechanisms included the use of hierarchical integrity policy augmented by "trust" and "integrity categories" in Gemini GEMSOS and VAX VMS/SE, the table-based approaches of LOCK and RACF, and the "capability" approach of Paul Karger's design and the RSRE SMITE machine. These TCBs ranged from paper designs to off-the shelf systems, and spanned the formal assurance ranges of the Trusted Computer System Evaluation Criteria (TCSEC) from C2 to A1 and beyond. Thus we were able to point to actual and potential systems which would withstand only unsophisticated attacks (those at the C2 level, with corresponding lower cost) to those which were designed to withstand highly technical assaults (those at the A1 level and beyond, with corresponding higher cost). We were satisfied that the cost/benefit tradeoff space was sufficiently populated, and accordingly declared the construction and assurance of an "E-rule" enforcement mechanism to be a solved problem.

In effect, we concluded that the Clark and Wilson requirements can be factored into two classes of simpler requirements: those concerned with access control (i.e., prevention of unauthorized access to CDIs), and those concerned with the preservation of the consistency of CDIs. The access control requirements, we found, can be met with appropriately high assurance by current or planned trusted systems. The remainder of our report restricts itself to the topic of how assurance might be gained that an application subsystem, supported by adequate access control enforcement, might be assured to preserve the integrity of the data it manages.

Having reached this conclusion, we then focused our attention upon the applications subsystem and the "C-rules."

2.3 Decomposition into Functional Elements

Concentrating now on the applications subsystem, the next step is to decompose it into elements that fit into Clark and Wilson's framework: TPs, IVPs, CDIs and UDIs. For each element a specification is developed that defines the element sufficiently to allow it to be constructed (detailed design and coding) as an independent unit of work. This detailed specification is derived from the overall specification produced in the second step, by partitioning those requirements, adding detail to them, and allocating them to the functional elements defined by the decomposition process. Essential to this process is the writing of an interface specification which describes the interactions between the elements.

Once constructed, each element is examined (through either formal methods such as program proof or informal methods such as inspection and test) to insure, to the required degree of assurance, that the element corresponds to its detailed, "allocated" specification.

2.4 Composition and Subsystem-level Assurance

A set of elements (TPs, IVPs, UDIs, CDIs) that perform adequately to their individual specifications is then composed into successively larger units and steps (generally testing) are taken to gain assurance that the assemblage both corresponds to its overall specification and exhibits the required integrity properties.

We wish to stress that this step is the highest-risk step in any system development activity and the point at which such activities most commonly collapse. The standard failure mode for software projects is to have a collection of elements, each of which works in isolation, but which cannot be made to work together. Historically, successful projects are those which have been careful to decompose the system and arrange the order of

composition or "integration" so that it proceeds in small and controlled increments. The worst situation is that of so-called "big bang" integration, in which all or a major subset of the elements are assembled at once; the attempts to get the resulting system to work invariably end in the explosion which gives the approach its name.

3. Doubts and Challenges

3.1 Demonstration of Invariant Properties

In the case of security and integrity properties for computer information systems, we noted two motives for putting forth a definition. The first motive is that the definition represents a universal truth, which places the definer in the attractive position of being the conduit for received wisdom. The second motive is more humble, and can be informally described as "truth in advertising:" offering to a potential user a more precise definition of the sense in which one's system "is secure" or "has integrity."

We stress that in the case of security and integrity properties the content of the definitions is much less important than their clarity. All computer information systems are embedded in a larger context in which security and integrity measures (e.g., restriction of physical access) can be taken. A system that exhibits a clearly defined but inadequate security or integrity property may still be safe in context, because the inadequacy of the system is evident and other steps may be taken to compensate for it. The real danger lies in illusion: security or integrity properties whose definitions are vague, inflated, or both. In these cases the user will be lulled into that most dangerous of states: a false sense of security.

In the discussion to follow, we treat the properties outlined in Clark and Wilson's paper with the second motive in mind: clarity and openness with regard to the implications of the rules. In this light we first examined the rules themselves to see if they represented an correct and appropriate definition of that which we individually visualized as "integrity."

3.1.1 Correctness

Being an assurance group, we defined "correct" as "corresponding to the definition of some higher authority," as in "a correct program is one which corresponds to its specification." We noted the difference between the confidentiality properties defined in the TCSEC, which are derived from a highly codified environment of Presidential Directives and Department of Defense regulations, and the properties proposed by Clark and Wilson, which come from a much less formal, "common law" kind of authority.

3.1.1.1 Doubt: Correctness

We doubted that an initial version of a policy would properly distill and abstract the intent of the diverse authorities which apply to the general area of data integrity, and saw no credible argument at this time that the Clark and Wilson rules are "correct" in any widely-understood sense.

3.1.1.2 Challenge: Collect Policy Statements

One way to produce such an argument is to find specific examples, similar to the example of double-entry bookkeeping advanced in the paper, that are mandated by some authority, and can be represented as instantiations of the rules proposed by Clark and Wilson.

We therefore recommend that representative examples of directives for information integrity be sought out and Clark and Wilson's rules be examined in their light. Examples of such directives are the Generally Accepted Accounting Practices, the Code of Federal Regulations, the Uniform Commercial Code, various stock exchange regulations on the reporting of earnings, Federal Aviation Regulations, OSHA regulations, and military directives.

In addition, this exercise would build an authority base for a national consensus as to the requirements for integrity in both commercial and military practice. The result should be a statement of accumulated policy similar to Chapter 7 of the Orange Book.

3.1.2 Appropriateness

We noted two definitions of "appropriate." The first was appropriateness to the applications area, i.e., applicability to a set of subsystems of interest. The second was appropriateness to the assurance task, in particular the crucial first step of demonstrating that one's system is not fundamentally unsound.

With regard to applicability, we observed that there exist numerous applications subsystems whose informally specified integrity properties correspond to those defined by Clark and Wilson. We further note that these applications subsystems exist in more areas than just commercial information processing, such as quality engineering and configuration management, maintaining the internal integrity of a system whose main objective is confidentiality (such as a TCB or a communications security system), and performing command authentication in a military environment. We accordingly view Clark and Wilson's characterization of their rules as providing a "commercial security policy" as being excessively narrow.

3.1.2.1 Doubt: Completeness

On the other hand, we observed that in order for the rules to be appropriate they must be complete. Our first doubt with regard to appropriateness was raised when we noted that Clark and Wilson did not mention the need for something as basic as a signature mechanism. As a result, applications such as electronic mail, the growing field of Electronic Document Interchange (whereby corporations can exchange invoices, purchase orders, etc. electronically), and other similar applications are omitted from their proposed definition of integrity. It appeared to us that Clark and Wilson had been unduly influenced by existing batch processing accounting programs and practices, and that they made relatively little attempt to abstract the commercial interchange of conventional paper documents and signatures into the digital world.

In particular, we noted that Clark and Wilson seem to assume a single accrediting authority, such as in some large corporation which certifies its own internal accounting system. But what if two corporations wish to exchange purchase orders, invoices, and ultimately credits and debits via Electronic Funds Transaction systems? Who then is the ultimate accrediting authority? This appeared to us to be an instance where a scheme such as that proposed by Clark and Wilson, which rests on a notion of centralized mediation, could turn out to be completely inadequate.

3.1.2.2 Challenge: Decentralized Control

Our challenge in this area is to demonstrate that the Clark and Wilson rules are appropriate in an environment in which control is decentralized. In particular, the Clark and Wilson formulation should be examined in the light of transactions between peers, such as separate corporations, which do not share a common accrediting authority.

3.1.2.3 Doubt: Consistency

It was with regard to appropriateness to the first assurance step that our serious doubts first began to be raised. If a set of rules is to provide a useful basis for assurance of an invariant property, they must be clearly defined and internally consistent. We found the definition of an IVP ambiguous with regard to purpose, in that it was not clear (nor resolved in the plenary sessions) whether the IVP was to check internal consistency, external consistency, or both. We were also uncertain about logging requirement of rule C4, in which the purpose of the log (accountability, backup, or both) was not clear. Again, this ambiguity was not resolved in the plenary session. We also felt rule E2 was possibly inconsistent with other rules. Its wording strongly implies the binding of individual names to programs

while other discussions indicated that it was the organizational role (a derivative value) that should be the determinant of who may initiate a given TP.

While we were not able to point out internal inconsistencies in the short time available, we noted areas of overlap in the rules that we felt could lead to such inconsistencies. We questioned the need for rule C1 and the need to distinguish IVPs from TPs at this high level of abstraction. Likewise, we felt that rule C5 was subsumed by C2, and again questioned the need for the distinction between a UDI and a CDI, or indeed the ability to state criteria which would enable us to distinguish one from another. We felt that rule E1 was subsumed by rule E2, and that rule E4 was an instance of rule C3 applied to the enforcement mechanism. The latter case raises the interesting question of circularity in the definitions, which we will touch upon in a later section.

3.1.2.4 Challenge: Formal Analysis

Our challenge to resolve these doubts is to recast the rules in some notation more formal than English and demonstrate internal consistency.

3.1.2.5 Doubt: Open-Endedness

Our second area of doubt centered on the feasibility of performing the crucial first assurance step at all in actual practice. We noted that central to the step was the striking of an "integrity perimeter" which bounded the portion of the applications subsystem to be reasoned about. That is, at all stages of the assurance process, including the first, it must be absolutely clear exactly which entities are within the system being reasoned about, and which are not: the "integrity perimeter" separates the two. This notion is crucial.

For example, a system that enforces the rules given in the paper, but does not treat the results presented to human users as protected data items through the entire path from generation to presentation to users, could not, in our opinion, be said to preserve integrity in any meaningful sense. It does not matter how carefully the internal representation of the data is kept consistent, if the programs presenting the data are not certified. Uncertified display programs provide no assurance that the data presented to users (and upon which their decisions were based) are accurate copies of the consistent internal data items.

A well-defined perimeter is necessary both for clarity of definition ("these are the things we protect and these are the things that protection depends upon") and to keep the assurance task within bounds.

We observed that Clark and Wilson's rules have a strong "embedded" flavor to them, in that the properties they define often apply to the interaction between the applications subsystem and the larger context in which it operates. This in turn implies that certain crucial operations will be outside any feasible integrity perimeter and therefore cannot be subjected to any form of a priori assurance. The rules rest on the fundamental principles of redundant integrity verification (IVPs, as set forth in rule C1) and separation of duties (rule C2). We foresee circumstances in which little or no assurance can be given that these principles are actually enforced by the system, because the enforcement mechanisms are outside the system's control. An example in the case of (one definition of) an IVP is that of the letters sent by a bank's auditors to the bank's customers, asking that they verify the balances in their accounts.

3.1.2.6: Challenge: A Worked Example

Our challenge then is to pick that oft-cited instance, a "real world" system with all its asymmetries and ad hoc features, and demonstrate that for at least one such system a satisfactory perimeter can be drawn.

3.2 Factoring and Allocation of Requirements

3.2.1 Doubts: Manageability of the Assurance Task

Our first doubt in this area had to do with the linkage between a global property such as separation of duties and the structure of the system. In particular, we noted that many "real world" systems have rules which, when applied to Clark and Wilson's structure of TPs, IVPs, CDIs, and UDIs, will yield a dynamic structure. In particular, one is able to cite cases in which separation of duty is invoked based on the value of the CDI (e.g., one signature on checks under $1000, two above that). The structure itself is then a function of the values being processed, and may not "stand still" to be defined, decomposed, and have requirements allocated to its elements.

Our second doubt was that the structure may change not because of linkage between CDI values and the configuration of TPs and IVPs, but because the integrity perimeter of necessity includes part of a changing management context. Of particular concern here is the definition of roles which in turn support the crucial concept of separation of duties. Organizations reorganize, and an invariant property which is a function of an organization chart may not be very invariant.

Our third doubt was that the structure of the system may be stable but may be too complex to be understood. We were informed of the LOCK experience, in which structures very similar to those

proposed by Clark and Wilson are used to maintain internal integrity of a TCB, and in which even very simple applications subsystems yield large numbers of TPs and CDIs. We also observed that Clark and Wilson's formulation, like earlier models, omits the crucial topic of synchronization and atomicity of operations. The inclusion of mechanisms in these areas (e.g., semaphores, message queues, and protocols) also tends to increase the number of TPs and CDIs in the structure. Furthermore, these "support TP/CDIs" are critical to the integrity by the system, and therefore cannot be ignored by the assurance task.

We noted that the requirement for separation of duty imposes a subtle requirement on the system with respect to the identification of users. In particular, the identification system, if it uses the notion of "groups," must 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 TPs. Otherwise a single individual could appear to be two different people, and thus avoid the separation of duty exclusion.

We also had doubts about the practicality of the IVP concept. We noted that the IVPs which check internal consistency of CDIs would most likely be constructed to a mirror image of the specification of the TP which produced the CDI. Given the rate of occurrence of specification errors, and the characteristic that the vast majority of them are errors of omission, we question whether the redundancy implied by the IVP concept is real or illusory.

3.2.2 Challenge: Continue the Worked Example

Our challenge to resolve the above doubts is to carry out the decomposition of the "real world" system proposed in step one, and to do so to a level of detail which included the synchronization mechanisms, and to demonstrate that the IVPs in the system do indeed verify the integrity of the data they examine.

3.4 Composition and Subsystem-level Assurance

This was the step in the assurance process where the group had its deepest doubts, which was discouraging because of the previously stated observation that it is the step in the overall development process that carries the highest risk.

3.4.1 Doubts: The Inductive Hypothesis

We severely question the ability, either formally or informally, to carry out the induction described by Clark and Wilson. We noted the extreme vagueness of the definition of "valid" and "well formed" in the paper, and doubt that there exists a non-

trivial definition of a "well-formed transformation" such that a sequence of well-formed transformations is itself well-formed.

For example, one definition of 'valid' for accounting systems, namely, 'the books are balanced' was used extensively throughout the paper and the plenary sessions. This definition, indeed, would appear to lead to a notion of validity that might be preserved if the rules were enforced. However, if that were the only meaning of 'valid' preserved by the application subsystem as an invariant, there would seem little reason to call the system a "high-integrity system:" it would be possible for users to corrupt the accounts in arbitrary ways, provided that their actions always left the books balanced.

It should be clear that the relevance of the Clark and Wilson proposal depends upon the existence of a great many application invariants that have the desired inductive property: if every transaction preserves some local invariant, a corresponding global and persistent invariant is also preserved. The informal induction presented in the paper appeared to us to use the word 'valid' in an ambiguous way: 'validity' for a particular TP seems to involve the preservation of the local consistency of the CDIs it operates on, while 'validity' in the conclusion seems to imply the preservation of global consistency for all of the CDIs taken together. It was not clear to us that a large set of useful invariants (i.e. those for which preservation of validity locally implies preservation of validity globally) can be defined, and we issue a challenge that such a set be described.

We note that the above problem exists in other areas: the confidentiality property exhibited by a computer network is often substantially weaker than the properties exhibited by its components, and the reliability of an assemblage of parts is generally far below the reliability of an individual part.

If our doubt in this area is valid, then there exists an upper limit to the degree of assurance that may be associated with an applications subsystem as an entity, and that degree of assurance may be unacceptably low in that no useful conclusions about the subsystem can be made. In such cases, then, even the minimal requirement that the security policy enforced by the system be clearly stated is not met.

Another doubt dealt with circularities in the dependencies between parts of a system. We have noted previously the necessity of incremental composition ("integration") and assurance. To perform such incremental operations requires a system design that can be "unwrapped" so that elements can be added to the whole in a sensible order. In particular, elements upon which other elements depend must be incorporated early and assurance about them gained in context.

Circularities are pathological cases in the overall structure in which element A depends for its correctness on the correct behavior of element B and element B likewise depends on element A. Circularities are pathological in that they force the development team into variations of "big bang" integration; in this case, the simultaneous incorporation of A and B into the system. From the point of view of assurance, of course, the assurance steps (however informal) must be applied to A and B together as there is no way to gain evidence that either is correct without considering the other.

We noted the potential for circularities in the rules themselves because of the E4/C3 relationship. Another example of a possibly inherent circularity is that of the user authentication data base, which is a CDI maintained by a TP whose correctness depends on the user authentication data base. We also noted that the path of the circularity may go outside the integrity perimeter and back in, because essential operations (such as a basic step in one alternative definition of IVP) can take place outside any feasible integrity perimeter. We also note that circularities are fatal to the inductive verification proposed by Clark and Wilson, in that they lead to indefinite recursive descent.

3.4.2 Challenge: A Rigorous Demonstration

Our challenge in this regard is to demonstrate that the induction is indeed possible, by taking a representation of even a trivial system in an appropriate formal notation, performing the induction, and submitting the result to the social process. An inductive proof of the overall integrity of an applications subsystem which had been checked by the Boyer-Moore theorem prover would be a convincing demonstration.

4. Summary and Conclusion

We end our report with a question, two admonitions, and a concluding observation.

The question addresses the issue raised often in the plenary sessions, to wit, what the economic value of all this is, and forms a final challenge: If one has a mechanism which enforces the E-rules to some degree of assurance, and one has an applications subsystem whose elements all exhibit Clark and Wilson's properties to varying degrees of assurance, and this system is to protect assets of a known value, then how does one estimate the insurance premium to cover loss?

The first admonition is to repeat the theme of the challenges that run through this report: work some examples. The history of computer science is replete with proposals for programming languages, system structuring techniques, notations, and assurance technologies which seemed plausible and even desirable

when described at the level of detail used by Clark and Wilson, but which collapsed utterly under the stress of actual use. In particular, we admonish the various organizations which are considering the establishment of standards, criteria, and certification mechanisms not to do so until considerable experience has been gained in the implications of these rules.

The second admonition is not to treat assurance cavalierly or as an afterthought. If one omits orderly and well-defined assurance steps, then one runs the considerable risk of providing just illusory security. Illusory security is adequate only if one's adversaries share in the illusion; if they do not, then it is worse than no security at all.

Our concluding observation is that the Clark and Wilson paper provides a useful first step toward codifying practices that have been informally incorporated in manual and automated systems for years. We feel, however, that a substantial amount of research and definitional effort must be expended before these rules can be used as the basis for any orderly or meaningful assurance activity.

5.0 REPORT OF THE WIPCIS
WORKING GROUP ON GRANULARITY AND FUNCTIONS

Dorothy Denning
Cheryl Helsing
Paul Karger
Kurt Meiser
Rolf Moulton
William Murray
Simon Wiseman

INTRODUCTION

In addition to sharing direct work on the Clark-Wilson Model, this working group was asked to address those issues relating to the granularity of Tranformation Procedures [TPs] and Constrained Data Items [CDIs]. Specifically, we were asked to provide guidelines as to the size of data objects that would provide appropriate separation of duties while not imposing an undue administrative burden. We were also asked to deal with the functions all TPs could be expected to perform.

Granularity

The extent, scope, degree or effect of a control must be appropriate to the intended application. In a security context, "The test of granularity requires that the size of the resource to be controlled be small enough to constitute an acceptable risk." [Data Security Controls and Procedures, IBM Corporation, March 1977, G320-5649] In general, the more granular the control or the smaller the object controlled, the lower the risk. However, there is a limit: there is a point at which increased granularity increases both the level of administrative effort and the level of complexity. The first can result in inaccurate or untimely rules, while the second can mask error or malice.

Functions

The group was also asked to comment upon what functions should be expected of all TPs. In a system that is composed exclusively of TPs, there might be some functions that all TPs would be expected to perform. Examples suggested, and addressed later, included parsing, consistency checking and logging. If these functions were sufficiently general, then each might simply call a generalized TP to perform it; if they were TP specific, then each would have to call a function to perform it.

Ownership

Finally the working group was asked to look at the issue of ownership of TPs and CDIs.

RESPONSE TO THE CLARK-WILSON MODEL

The group spent it's first four hour meeting, discussing the Clark-Wilson model and responding to the questions about the model that had been posed to them.

General

In general, the group strongly supported the ideas advanced by Clark and Wilson. They felt that the paper represented a great deal of progress on a "difficult problem." They did not identify any invalidating flaws or deficiencies. They recommended some rewording of the rules and those are included in an appendix to this report. Nonetheless, there was a strong sense of caution and that more testing of the ideas is indicated.

Issue of a Separate Integrity Model

The group expressed a strong preference for a single computer security model embracing confidentiality, integrity and continuity as peer functions. They felt that earlier models had concentrated too heavily on confidentiality. Previous models (ie: Bell and LaPadula) concentrated on confidentiality because integrity was just not understood well. At the time they were developed, both integrity and confidentiality were equally high priorities. However, a solution to confidentiality was found then (early '70s), but Clark and Wilson seems to be the first good model for integrity.

On the other hand, there seemed to be a concensus that it was not productive to consider each in isolation from the others. It was felt that the same mechanisms could be useful in achieving each. In addition, since one objective might be achieved at the expense of the others, then considering them independently might be dangerous.

The group felt that the issue was completeness of whatever model or set of objectives might be used. They rejected the proposition that the requirements or desires of military systems were substantively different from those in commercial systems, as might have been inferred from reading the Clark-Wilson paper. Likewise, they did not believe that there was an issue around whether particular controls were "mandatory" or "discretionary." Instead the issue was seen as whether there were a sufficient number of people involved and when the rules are bound. For example, the necessary and accepted concept of "mandatory" as used in the national security context, can be described as a special case of separation of duties bound at a very early time.

Similarity to the Confidentiality Model

While noting that the same control primitives might be useful for both integrity and confidentiality, they indicated that a new model would be required to relate the access control primitives to integrity.

The group took note of a working paper, "Using Mandatory Integrity to Enforce 'Commercial' Security" by T.M.P. Lee. This paper suggested that "when properly applied" the mechanisms described in the Trusted Computer System Evaluation Criteria would be effective in enforcing the Clark-Wilson model. While accepting that this was likely, the group still indicated that the model used should closely match the objective. The user should not be required to invent or force a fit. Force-fitting the Clark-Wilson model into the framework of the Biba integrity model (so as to use DoD-type access control mechanisms), while possible (as shown by Lee), is not particularly useful or enlightening. Similarly, the group felt that the mechanisms suggested by Clark-Wilson might be sufficient to enforce the DoD Security Policy but that an unnecessarily large amount of work might be required to do so.

Certification vs. Enforcement

The group registered some concern about the distinction drawn by Clark-Wilson between those rules that were labelled as "C" and those labelled "E." It seemed as though there was an implicit assumption that the former could be attested to by competent authority but not enforced by the system. Yet, the group found two "C" rules which they felt might in time be enforced by the system. They suggested that the distinction might be based in part upon Clark and Wilson's view of what was practical and that it might discourage attempts at enforcement. The group felt that while certification is both indicated and necessary, that its use should not be mandated where enforcement might also be an equally appropriate choice. Certification was seen as one tool among others, and not to be automatically preferred. (A subsequent discussion suggested that perhaps the distinction was between those rules that were application specific (C) and those that were specific only to the system (E)).

GRANULARITY

The working group concluded that granularity of TPs and CDIs was ultimately an application issue. While it is difficult to conclude exactly what granularity is required except in the context of a particular application, the group identified useful guidance.

Single Function

The group concluded that TPs should be limited to a single well-defined function or be composed only of like or related functions. While too many small particles may be messy and administratively burdensome, in computer systems, it is far easier to group things than it is to divide them.

Another test that was agreed to was that a manager should never be forced to grant privilege A simply for the purpose of conferring B. A and B should be separate units. In that case, sets A-B could be formed as could A-A and B-B. On the other hand if AB was the smallest named unit, that would fail the test.

Separation of Duties

The group took note of the generally accepted standards of good business practice for separation of duties (see position paper by W. H. Murray, attached). These practices require that at a minimum managers must be able to separate the following:

* origination from approval

* creation from maintenance

* procedure from data

* assets from records

Obvious Intent

The TP should be small enough that it is obvious in its intent. The larger the object, the more difficult it is to comprehend. When administering the TP, the manager should be able to predict its effect and the effect of delegating authority over it from the information in its name and the name of the TPs to which it is to be joined. However, Clark-Wilson requires [rule C3] that the set of TPs delegated to any single user must be certified to meet appropriate separation of duties.

Explicit Conflicts

In a position paper submitted to the Workshop, this reporter suggested that the list of conflicts should be made explicit and enforced by the system. When the number of data objects reaches the hundreds, and the activity to access control lists is high, then "certifying" the absence of conflicts would rapidly become overwhelming.

Similar in Size to Traditional Data Items

The group suggested that TPs should be similar in size to objects that managers are accustomed to dealing with. Examples submitted included: business transactions, pages, forms, screens, and documents.

IMPLICATIONS

It seems clear that in many business applications, both TPs and CDIs will be smaller than the data objects that are named or controlled by most operating systems. They are likely to be more of the size and type that are most often associated with such operating system extensions as database managers, transaction monitors and communication monitors. However, the Clark-Wilson model permits these extensions, as well as application processes to be described as TPs.

There is no clear delineation in Clark-Wilson between that function that must be provided by the "operating system," its extensions and subsystems, or the application. There is an assumption that any contained and orderly function will be trusted for that which it has been assured to do. Since these processes can be arbitrarily granular, and since arbitrarily limited and simple processes can always be assured to perform their function, (e.g., most primitive computer instructions can be demonstrated, by inspection or exhaustion to do what is intended) then any required degree of confidence can be achieved.

However, there are functions that the "operating system" must provide to support Clark-Wilson. These center around the notion of protected subsystems to ensure that TPs can do their jobs without inteference. Examples of operating systems with such protected subsystems include IBM's System/38, Honeywell's Secure Ada Target (now called LOCK), and other capability-based systems.

On the other hand, it appears that, unlike the claims made for Bell-LaPadula, no single process can be relied upon to enforce Clark-Wilson. The Clark-Wilson model ultimately depends on the correct implementation of the applications specific TPs. In a community that has traditionally held that no applications, and only a few operating systems are trustworthy, this may account for some of the reservations that were expressed about the model.

MANDATORY FUNCTIONS OF TPS

Validate Inputs

Clark-Wilson requires that "TPs which accept UDIs must be certified to perform only valid transformations, or else no

transformations, for any value of the UDI. The TP should take the UDI to a CDI or reject it." Specifically, this must include validating formats and codes. It also includes proper encoding and validation of that encoding.

Logging

Logging of what happened is a key component of Clark-Wilson. Reconciliation of logs to the environment is a key mechanism for ensuring agreement with the external environment. Clark-Wilson requires that "TPs must be certified to write to an append-only CDI all information necessary to permit the nature of the operation to be reconstructed [rule C4]. Our group agrees that writing to a log is indicated. However they questioned the wording of the rule. First, we were concerned about the idea of "append-only CDI." While we recognize the requirement to make the log tamperproof, we could not agree that "append-only" was the only way to make the log safe. Second, the group felt that the ability to read the log was important; they wanted to caution against wording that might appear to restrict it. Third, we felt that the integrity of the system is improved when there are multiple independent logs; we want to caution against too much reliance on the log kept by the TP and encourage the use of other logs.

Nonetheless, there are significant application events and content that are visible only to the TP. It must be held accountable for the logging of those events. It should also be the logging process of last resort.

Alarms

Similarly, the TP must be the responsible process for alerting management to those events which only it can see.

Termination Report

Finally, the TP must report the nature of its termination. It must indicate in the log whether it went to normal completion, was terminated by the user, terminated itself because it could not leave all CDIs in a valid final state, or was terminated by the environment.

OWNERSHIP

The working group was asked to deal with the issue of ownership. We used the definition "the exclusive right to use." The "owner" is the individual that exercises the organization's proprietary rights over the data object. In access control, the owner is the individual that exercises GRANT or PERMIT privileges over a data object. These privileges are similar to those associated with Clark-Wilson rules E1, E2 and C2.

6.0 REPORT OF THE WIPCIS
WORKING GROUP ON IDENTITY VERIFICATION

Tom Berson (Leader)
Peter Capek
Jim Schweitzer
Clark Weissman

INTRODUCTION

Every reader of this report has at some time verified his or her identity to a computer system. Entry of a userid and password in response to computer prompting is the almost universal model for this simple but essential act.

Although the (userid, password) model is nearly universal, it is often unsatisfactory from an information security standpoint. Passwords may be intentionally given away, accidently revealed, or even successfully guessed. Spectacular computer crimes of fact (Jeremy Rifkin at Security Pacific Bank) and of fiction (the film War Games) are based upon the assumption by the attacker of a fraudulent identity.

Verification of a user's identity supports two key security requirements in an information processing system. First, control. Access control mechanisms of all sorts (e.g. RACF and Multics' ACLs) are parametric on the user's identity, thereby providing means for differentiated user authorization and permission. Second, accountability. Audit and investigation rest upon correct linking of information system events to the users who prompted those events.

Our working group was asked to consider identity verification in the spirit of WIPCIS. We found that the demand for high integrity information processing, as represented by the Clark and Wilson paper, does have an effect on requirements for identity verification. Two strong themes of WIPCIS were: 1) separation of duties and 2) reality checking. Both of these have found their way into this report.

Careful inspection of Clark and Wilson shows that while identity verification is required (rule E3 is totally devoted to the subject) it is not modeled. From our working group's parochial point of view it seems fair to summarize C-W as simply stating a requirement for high integrity identity authentication.

The body of this report is a set of statements we have boldly called "Principles of High Integrity Authentication." Each represents the conclusion of our discussion on one aspect of authentication. Many are accompanied by a capsule summary of the discussion.

Some of these principles may seem to be "simply" common sense. We think that information system security is a field where common sense can be applied. We expect that our articulation and compilation of these principles will prove useful to system designers and evaluators.

We think that these principles are correct, but have no evidence that they are complete. We invite readers to send us their comments and to suggest additions.

For purposes of organization we view the identity verification subsystem as composed of three components: the user, the communications path, and the authentication server. Principles have been assigned to the component which they most effect. There is a fourth category, system issues, for aspects of the problem which are not easily assigned to a single component.

Note that we talk in terms of authentication of a user to a system. Some applications may also require authentication of a system to a user (mutual suspicion). The same principles apply.

Further, some applications may require the authentication of system parts (e.g. portions of a distributed TCB, smart crypto controllers) to one another. The same principles still apply, although the wording may require de-personification.

DEFINITIONS

User -- the person whose identity is to be verified. For example, the flesh and blood accounts payable clerk.

User name -- The name by which a user is generally known to other people. For example, "Jubilation T. Cornpone."

Id -- the name by which a user is known to a computer system. For example, "123-45-6789", or "jubi".

Authenticator -- evidence which a user provides as a bona fide of his/her identity. For example, a password.

Authentication server -- the mechanism that performs registration of users, authentication of users, and maintains the authentication data base (adb). The authentication server may be a process, a dedicated host on a network, or something in between.

PRINCIPLES OF HIGH INTEGRITY AUTHENTICATION

-- USER --

1. Every id must be unique within a reality domain (e.g. a

single administration) [see the Reality Working Group's report for more on reality domains] and within the retention period of the audit log.

If an id were not unique in the space of a reality domain and over the time of an audit log then control decisions and efforts to assign accountability would suffer from ambiguity.

2. Every id must have a single (human) user assigned.

Again, we seek to avoid ambiguity. This rule forbids "group ids." When users in similar functions must be granted similar access this should be arranged through the access control mechanism.

3. A user may have multiple ids, but not within a single reality domain.

Our goal here is to preclude aliases by which a user might subvert a separation of duties policy. We note that there appears to be a tension between separation of duties and those privacy mechanisms which provide a user with multiple unlinkable identities so that his various actions cannot be correlated. For the moment, we come down on the side of supporting separation of duties.

4. Multiple roles should be handled by a "change roles" TP external to authentication.

Some people fall to the temptation of assigning an individual user multiple identities as a way to acknowledge that a user may have multiple roles, each with its own access. This could lead to failure of a separation of duties policy. We believe that the multiple roles should be explicitly acknowledged by the access control mechanism.

5. Every id should have a single authenticator associated at any particular time.

This is a simplifying principle. Cancellation or expiration of an authenticator becomes procedurally equivalent to revocation of access from an id. The principle may be ignored in cases where the simplification is unwanted, such as the temporal overlap of encryption keys or n-party authenticators used in "no lone zones."

6. The authenticator is a function of what the user knows (e.g. a password) or what the user has (e.g. a badge) or what the user is (e.g. a fingerprint) or what the user can do (e.g. a voiceprint).

We expect that authenticators will increasingly come to be a mix of all four.

7. Different sorts of authenticators have different characteristics in terms of strength, convenience, transferability to other users, etc.

8. When authentication succeeds (the user's identity is verified), positive feedback should be given that provides information for user self-audit (e.g. date and time of last login). The feedback should be given in user-relevant terms.

It does not seem of much use to tell a person in the Pacific time zone that he last logged in on 87241 at 0902 GMT.

9. When authentication fails care must be taken in whatever negative feedback is given to avoid revealing information of use to attackers.

Specifically, care must be taken to prevent independent attacks on the id and the authenticator.

10. The authenticator must be changed periodically.

We think that the period for which an authenticator is of value should be related to the exposure which the authenticator has suffered. The objective is to prevent spoof and playback attacks. Some network applications may call for change of password as often as after a single use (automatic mechanisms are available for assistance with this). Systems which allow users to invoke programs of their own manufacture or choice are so vulnerable to spoof or playback that reusable passwords are almost never a safe choice of authenticator.

11. Authentication may take place whenever needed.

We tend to think of authentication as being something that takes place at "login" time, but there is no reason why it should not take place at application entry, at transaction entry, or several times during a transaction. Or at any appropriate combination of these times.

12. After quiescent periods user may be required to re-authenticate.

This raises the interesting question: How "long" is an identity verification valid? The answer might be in terms of idle time, elapsed time since last authentication, number of transactions, value of transactions, etc. The answer to this question also depends, in part, on whether the system is operating synchronously or asynchronously at the application

layer. Asynchronous transactions must each carry evidence of their authenticity (and origin, and integrity).

13. Different transactions may require differing levels (quality) of identity verification. For example, access to the stock ticker may require only casual verification, while ordering a trade may be occasion for more severe scrutiny of the user's identity.

-- PATH --

14. Authentication messages should be viewed as a protocol suite.

We would like to subject the message exchange to analysis for correctness, liveness, recovery from error, etc. We would also like to view the end points of the protocol (i.e. the user (or an authentication engine in the user's possession) and the authentication server as communicating peers.

15. Authentication communication between user and authentication server must be protected against disclosure of authenticators,modification of messages, and insertion of messages.

This goal may be assisted by end-to-end encryption, by protected wirelines, by one-time passwords, etc. We want to prevent attacks on the authentication process from taking place via the communications path. The path is less observable than either the user or the server, and, therefore, more vulnerable. Good security engineering practice encourages attack in easily observed places.

Note that the terminal itself is part of the path.

-- SERVER --

We envision authentication services being provided by one or more "authentication servers" against which a few transactions are defined.

16. Authentication services should be realized as a set of TPs encapsulating a set of CDIs comprising an authentication data base (adb).

Strong typing is not sufficient. The TPs that control the adb CDIs must be subject to some sort of "certification" and operated in a security architecture to provide assurance against misuse of the adb.

17. An enrollment transaction adds to the adb a relation binding the user name, id, a reference authenticator, and enrolling authority.

The reference authenticator is the data to which the authenticator is compared. For example, (a non-reversible transform of) a password is compared to the reference authenticator. For another example, a freshly collected signature may be compared to a reference authenticator collected earlier. The reference authenticator may be supplied by the user (e.g. a bank signature card) or by a trusted party (e.g. the state) using a forgery resistant token (e.g. a driver's license).

The enrolling authority is the id of the person on whose authority the user is enrolled. This should be a person in a relevant line or command position. It should not be the enrollment administrator, who doesn't get paid enough to take the rap for an unauthorized enrollment. An example of separation of duty: one person authorized, another executes.

18. Privilege is granted and adjusted independently of enrollment (by the access control system, not by the authentication system).

This is another example of separation of duties: one set of people and TPs enrolls a user, a different set of people and TPs empowers the user.

19. The update transaction -- Adb data may be changed only by appropriate parties.

In password schemes, a user may be allowed to change his reference authenticator. In token schemes the reference authenticator is the token's secret identity. The user has beneficial use of this secret, but he is not a party to it, he cannot recover or duplicate it, and must not be allowed to change it. Most adb data should be changed only by the enrollment administrator.

20. The authentication transaction uses the adb and an exchange of messages to determine the validity of the user's claimed identifier.

This is the payoff -- a judgment by the authentication service on the validity of the user's claimed identity.

Care must be taken by the authentication server to protect itself from exhaustive attacks and playbacks.

Care must be taken by the user to protect himself from being duped into surrendering secrets to processes other than

legitimate authentication server processes. This was simple to do in closed systems. It is so difficult to do in open systems that the user is best not made a party to the secret.

Care must be taken by processes relying on the authentication service to prevent spoofing by a spurious authentication service, to prevent accidental or malicious modification of the authentication service's judgment between the time it is given and the time it is used, and that the judgment is used correctly.

21. The authentication transaction must be over-designed to avoid incomplete transactions, incomplete adb updates, exposure of passwords (e.g. must enter old password to activate the new password), etc.

The authentication server and its transactions are keys to the kingdom. They require special "trust" attention.

22. Enrolled users must be periodically reconciled to reality.

Yes, you have to go out and "find" each of your enrolled users. Are they still living, still employees, and still authorized to use the system?

--SYSTEM ISSUES--

DURESS. Some applications require that the identity verification system enable the user to pass a subliminal (not noticeable to an outside observer) message that the user is under duress. We observe that this is possible to provide, but seldom needed, and seldom used when provided. We recommend that if a duress facility is provided users be instructed that its use is optional.

TRUSTED DISTRIBUTION. It is often wise to authenticate the provenance and contents of an update to operating software or data. Cryptographic and protocol techniques are available to support this requirement. We believe that no WIPCIS working group discussed this.

Code updates may be thought of as "external" TPs, where the update modification is a transaction against a CDI called "the operating software" by an authorized user properly authenticated and authorized. Unlike the "internal" TPs, which have omnipresent hardware and software enforcement, these "external" TPs escape these enforcements and therefore require a greater level of human diligence. This is another arena for separation of duties.

DELEGATION. At one of our presentations to the plenary session a lively discussion took place about delegation, acceptance, revocation of delegation, and override. We think that delegation should be accommodated by an explicit transfer of access rather than by allowing the delegatee to operate under the delegator's identity. Revocation can then be handled as a properly authorized and executed "de-enrollment" transaction.

RESEARCH TOPICS

1. Relation between separation of duties and anonymity. Separation of duties drives the requirement for mononymity, (monoidentity would be better), of users. Privacy may require multinymity, in the sense that in order to prevent "big brother" type of attacks, a user may wish to use multiple linked untraceable unforgeable aliases [see Chaum].

2. Estimation of security problem leading to adaptive reaction. For example, Capek's login bandwidth choke, a scheme where failed authentication attempts are delayed in proportion to the number of previous failed authentication attempts involving either the id, the authenticator, or the access path.

3. Understanding the complex trade-offs between Type 1 (false acceptance) and Type 2 (false rejection) errors for different authentication schemes and for different security policies. For example, a requirement for very low Type 1 error rates will inevitably result in high Type 2 errors rates. In other words, the system is adjusted to refuse some authentic users so that it also will refuse most inauthentic users.

FURTHER READINGS IN IDENTITY VERIFICATION

David Chaum. Security without identification: Transaction systems to make Big Brother obsolete. Communications of the ACM 28(10):1030-1044, October 1985.

DoD 5200.28-STD. Department of Defense Trusted Computer System Evaluation Criteria. DoD Computer Security Center, Fort Meade, MD, December 1985. The "Orange Book."

DoD CSC-STD-002-85. Department of Defense Password Management Guideline. DoD Computer Security Center, Fort Meade, MD, April 1985. The "Green Book."

Charles Hemphill Jr. and John M. Hemphill. Security Procedures for Computer Systems, chapter 10. Dow Jones-Irwin, Inc., 1973.

Lance J. Hoffman. Modern Methods for Computer Security. Prentice Hall, Inc., 1977. See especially chapter 2, "Authentication."

David K. Hsiao, Douglas S. Kerr, and Stuart E. Madnick. Computer Security. ACM Monograph Series, Academic Press, Inc., 1979. See especially section 4.3. This reference contains a review of relevant articles through 1979.
Harry Katzan, Jr. Computer Data Security. Van Nostrand Reinhold Co., 1973. See especially Section 6.3, "The implementation of access management."

James Martin. Security, Accuracy and Privacy in Computer Systems. Prentice Hall, Inc., 1979. Chapter 11 contains a full description of identification by what you know, what you are, and what you carry. See also chapters 12 and 13.

W.H. Murray. End User Authentication. Ernst & Whinney, Cleveland, Ohio, September 1987.

National Bureau of Standards. Password Usage Standard. Federal Information Processing Standards Publication 112, U.S. Department of Commerce, National Bureau of Standards, May 1985.

James A. Schweitzer and Charles R. Symons. A Proposal for an Automated Logical Access Control Standard. In Proceedings of the 1984 IFIP Conference, Toronto (1984).

Raymond M. Wong, Thomas A. Berson, and Richard J. Feiertag. Polonius: An Identity Authentication System. In Proceedings of the 1985 Symposium on Security and Privacy, IEEE Computer Society Order Number 629 (1985).

Helen M. Wood. The Use of Passwords for Controlled Access to Computer Resources, NBS Special Publication 500-9, U.S. Department of Commerce, National Bureau of Standards, May 1979.

7.0 REPORT OF THE WIPCIS WORKING GROUP ON AUDITING

prepared by:
Peter D. Wild

November, 1987

Group Members:
Peter D. Wild, Leader
Peter Neumann
Dennis D. Steinauer

INTRODUCTION

Originally The Audit Group (the Group) was given only one Rule to consider:

C4: TPs write to Log.

All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

Very early on, in the discussions, the Group felt that it was important that the audit implications of all the Rules should be considered. It was on this basis that the Group discussions then continued and this also is reflected in the structure of this report.

We felt that the requirements for auditability within any system are very similar, if not identical, to the requirements of management and the user. This not only applies to the availability of information and data, produced by the system, but also the level of confidence which can be placed in the system in order that it will behave according to description. An audit will use the necessary data and information to compare the actual behavior of the system with expectations after considering whether those expectations are appropriate. An important implication of this statement is that we are not suggesting change or enhancement of the Clark Wilson Model solely for the purposes of auditability, but more in order that the precepts of good management control can be implemented and followed.

We felt that the following prerequisites for good management and control, and also auditability, could be stated as follows:

There must be unequivocal identification of users and processes within the system.

There must be clearly defined interfaces between all the components which comprise the system. This must define,

among others, the degree to which the component is secure and therefore its "right" to use an interface without compromising security. It was felt that there may be a parallel here with the "domains" which were mentioned and discussed by the Working Group on 'Agreement with the External Environment'.

Division of Duties, or Segregation of Duties, is an important concept discussed in the Model. The fundamental requirement for this concept to be understood, in the context of a particular organization, is the presence of an Organization Chart with the appropriate Job Descriptions for each Job Function. In many places the term 'user' is mentioned, in the Clark Wilson Model, and we felt it should be replaced with the term Job Function.

In order for any of the concepts, described in the Clark Wilson Model, to work and, perhaps more importantly, to be seen to be working, the foundation, upon which the application system operates, must be secure. In modern computer environments this is another way of saying that the Operating System environment must be secure and one definition of such a secure environment is a Trusted Computer Base (TCB) as specified in the 'Orange Book' (The Department of Defense, Trusted Computer System Evaluation Criteria). It can therefore be said that the Application System, as envisaged by the Clark Wilson Model becomes an extension of the Trusted Computer Base and may be called the Integrity Trusted Computer Base (ITCB). (Please see Diagram at end.)

The Individual Rules.

Here begins a summary of the comments made in respect of each of the individual rules. Our objectives were to clarify, not only our own understanding, but also that of others who may, in the future, read the Paper. We have, therefore, suggested a rephrasing where we considered it appropriate. We also felt that the sequence of the Rules was important and we have attempted to reflect this in the list that follows. We feel, however, that more consideration should be given to this as the Model matures. An example of this is the Rule C1 about IVPs, we think that this should be last if only because it presumes an understanding about CDIs and TPs.

C2: TPs Preserve Valid CDI State

Suggested Rewording:

Execution of a TP will only result in a valid CDI state.

Comments.

The execution of a TP is, of itself, a loggable event.

TPs and also IVPs are also CDIs with special characteristics.

The rule commentary, which mentions the role of the security officer, should be changed. The reference to the security officer is probably too narrow since the role of the security officer is one which is delegated by the owner or custodian of the data and the programs which operate upon it. The security officer's role is to ensure proper separation of duties during the process by which the TP is promoted to production status and the TP is then only made accessible to authorized Job Functions (i.e. users). This promotion process must also ensure that the nature of the access of the TP to the CDI (such as Read Only, Update, Delete and Create) is also appropriately authorized by the owner or custodian.

C4: All TPs Write to Log

Suggested Rewording:

The invocation of a TP is a loggable event.

Comments.

The original rule commentary appears to envisage the log being used only for the purposes of reconstructing the nature of the operation. This was felt to be too narrow since we were aware of other, very important, uses of a log. These would include:

Recovery of the data files.

Auditing of operations.

Investigation of operations and events, after the fact, as well as providing evidence of same.

System Monitoring for either Performance considerations or perhaps for Threats to the system.

Accountability and/or Billing requirements within the system.

It was noted that it is very likely that different data would be required for each of these purposes and it would be ideal if all the necessary data could be logged at one time, in one place, to meet these requirements. Concern was expressed as to the resulting volumes, but this would be likely to be minimised, in the future, with the maturation of new mass-storage technology such as CD ROM. What was of more concern, however,

that any particular purpose could be efficiently achieved.

We also stated that the integrity of the log data itself was at the very foundation of the Model. This data must be the most secure and attract the highest level of integrity in order that the application data could be recovered or investigated without any doubts as to its authority. Therefore the logging mechanism, itself, must be trusted. Another important point which was stated was that the retention periods for this logged data must be seriously considered, but we did not have the opportunity to define any guidelines in this respect.

C5 TPs validate UDI.

Suggested Rewording :

A UDI can be transformed to a CDI only through a valid TP.

Comments.

Since a TP, at an early point in its life, can be regarded as a UDI, the Group felt that it would be advantageous to use this type of mechanism to promote such a TP to production status. The TP to achieve this would probably have to be a part of the Trusted Computer Base. The Division of Duties over this process would, of course, be a critical part of the whole Model.

C1 IVP validates CDI state.

Suggested Rewording :

For every CDI, there must be a set of IVPs that ensures that the CDI is in a valid state at some defined point in time.

Comments.

We felt that the Clark Wilson paper should be further developed in the definition of all the components that comprise an IVP. This is because we felt that while an IVP has a computerized component, which in and of itself should be a TP - even though it would not change the CDI, there were also other essential clerical components to the succesful execution of an IVP. Also it is obviously possible that the execution of an IVP could lead to adjustments being made to computerized records. These adjustments must be made by either regular or special TPs. The effectiveness of an IVP is also directly dependent upon the Division of Duties exercised during that execution.

The Group also felt that the execution of an IVP was an important - and therefore - loggable event. We also felt that the quality and integrity of the CDI was enhanced by virtue of the execution of an IVP upon it. The converse of this is that a CDI

becomes increasingly questionable in its integrity and accuracy as time passes since the last IVP. It may therefore be relevant to time stamp the CDI as a result of the execution of a IVP upon it. It is also clear that the appropriate time period between IVPs on a CDI must be determined in relation to the importance of the CDI, and the non-execution of an IVP within the specified time period should be a reportable event.

An IVP is a part of the normal application system, it can also be identical - or very similar - to a procedure used by an internal or external auditor. Perhaps there is a case to be made for saying that an IVP, when executed by an auditor, has, in some way, more "value" since it could be argued that a greater degree of Division of Duties was observed during that execution.

E2 Users authorized to TP.

Comments.

As mentioned above the Paper makes reference to "Users". The Group felt that this word should be replaced, in many cases with Job Function, the concept being that individual users should only acquire access to CDIs, through approved TPs, by virtue of their Job Functions. The Job Functions, as mentioned above, are specified in Job Descriptions and the "map" of the Division of Duties is shown in the Organization Chart for that organization.

The Division of Duties over the whole process by which TPs are set up and users acquire access to them is obviously critical to the stature of the Integrity Trusted Computer Base (ITCB).

The Group also felt that there should be a series of IVPs to periodically check the integrity of these definitions of access to TPs and CDIs.

E3 Users must be authenticated.

Comments.

The Group felt that this is a function which must be performed by the Trusted Computer Base (TCB). The user must be authenticated before any opportunity is provided to invoke a TP.

The Group felt that the remaining rules which are listed below should be deleted for the reasons given.

E1 CDIs changed only by authorized TPs.

Comments.

This rule is subsumed by E2, and this was generally agreed by all Groups during the general discussion.

E4 Authorization Lists changed only by Security Officer.

Comments.

This was regarded as only a particular implementation example of Rule E2. It was seen as being an advantage to have a similar procedure as for normal TPs, since the Authorization Lists would have to be changed by a TP.

C3 Suitable separation of duties.

Comments.

This is another implementation example of Rule E2.

Other Issues and Questions.

The Group considered that it was important to define, more clearly, the relationship between the Clark Wilson Model and the Trusted Computer Base (TCB). We felt that the Clark Wilson Model could only successfully exist in an environment where the TCB was the foundation. Also, as mentioned above, some of the functionality described in the Clark Wilson Model could only successfuly operate at the TCB level. As part of an answer to this we provide a redrawing of the diagrammatical representation of the Model whch appears on Page 192 of the original Paper. Our objectives were not only to demonstrate this point but also to attempt to simplify the picture for clarity. (See Appendix 1 to this report.)

We felt that the implications of the Clark Wilson Model should be made clearer for the purposes of providing guidelines for the Systems Development process and also for the Vendors who are developing any type of product which operates in the future commercial systems environment. These products would not only include Systems Software such as operating systems and access control software but also application systems and such environments as Fourth Generation Languages. We also expressed a concern as to the difficulty of implementation of this Model in the light of current environments and also what is currently known about imminent developments such as OS/2 on the Personal Computer, and Systems Application Architecture which will provide a structured environment for running a system on almost any combination of microcomputer and mainframe. The early involvement of the developers of these environments would be highly

advantageous to the future of the Model.

Possible Research Projects.

The Group felt that there would be great benefit in encouraging Research Projects in the following areas:

The definition of the data necessary for logging in the light of all the possible uses which could be made of that data. An important part of this project would be the appropriate navigation methods available for the use of this data for a given purpose.

The use of the log data for audit purposes, particularly in the area of what has been referred to as "Audit Reduction". This is taken to mean how to reduce, what is presumed to be the high data volumes, into a form which is useful to the auditor to provide a necessary commentary on the quality of control and the integrity of the data.

Other Groups, apart from ours, felt that that much would be learned from the building of a prototype of the Model.

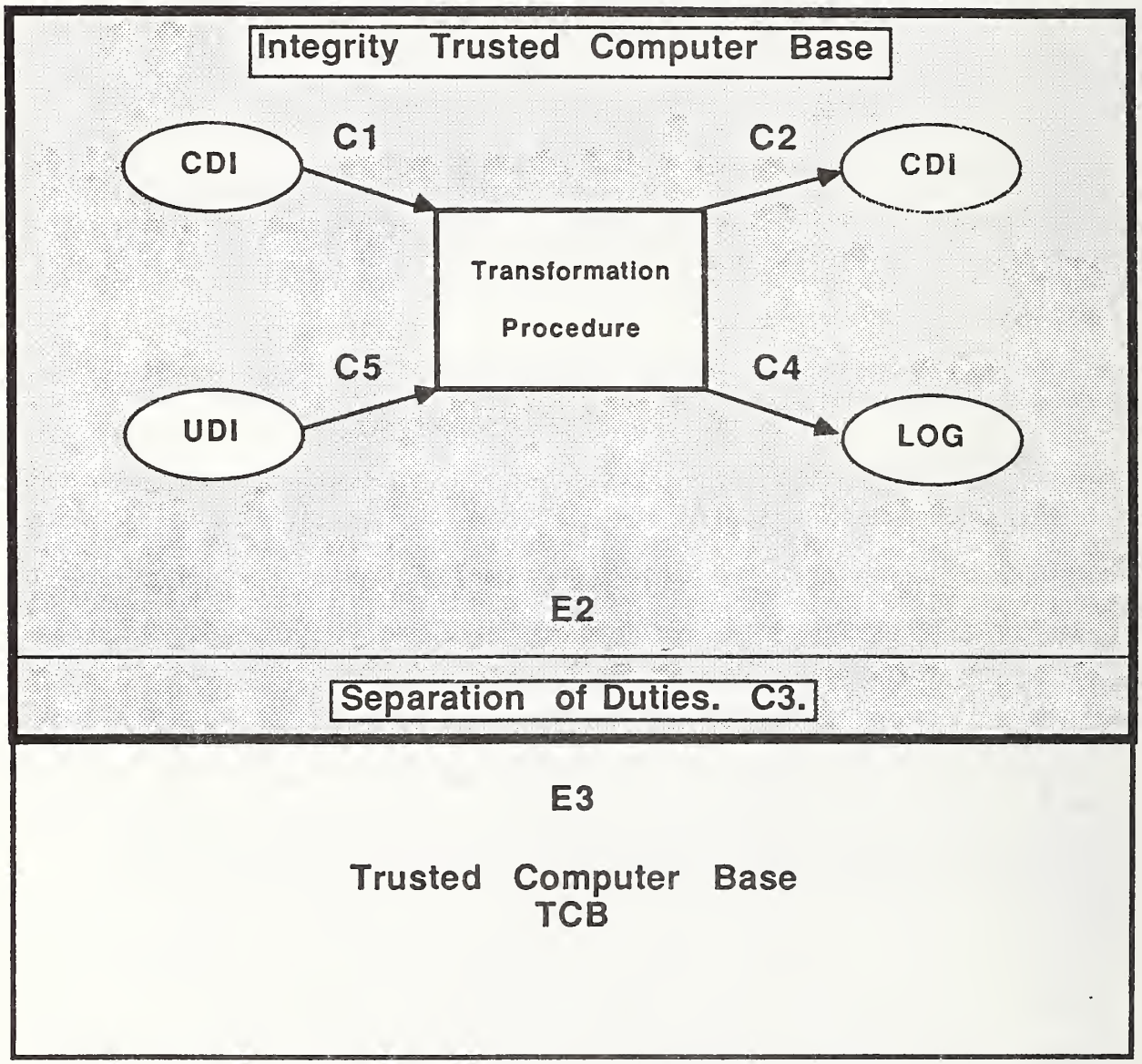
Conclusion

The whole Group unanimously felt that the opportunity to participate in the Workshop was an extremely valuable experience, and one that none of us would have liked to miss. We feel that some extremely useful progress has been made, and, at an absolute minimum, we are beginning to develop a useful vocabulary which can be used to discuss the problem. We would like to sincerely thank the organizers for the opportunity and would like to express an eagerness to continue our participation in the future.

On a personal note, as the Group Leader, I would like to sincerely thank the members of the Group, and Stuart Katzke who joined us for some of our discussions, for providing a very exciting forum and what I consider to be some very valuable ideas.

Appendix 1.

7.1 Suggested Redrawing:



7.2 Summary of System Integrity Rules.

8.0 REPORT OF THE WIPCIS WORKING GROUP ON CORRESPONDENCE OF A SYSTEM TO REALITY

by David R. Wilson

December 1, 1987

Introduction

The Clark-Wilson model, using Rule C1, called for an integrity verification procedure to occur periodically to test the integrity of the data. The integrity verification procedure really had two parts, a test of the internal consistency of the data, and a test of the data back to real world values. For example, an integrity verification procedure for an inventory system might include both an internal balancing procedure, and reviewing actual bin counts of items within bins within a storeroom. The Clark-Wilson model, in Rule C3, also called that adequate separation of duties be established as the primary mechanism by which external integrity could be assured. This workgroup was to examine the completeness of Rule C1 and C3, and to suggest some approaches to their implementation. The workgroup consisted of Paul Peters, John Fitzgerald, Marv Schaeffer, David Brewer, Donn Parker, David Bell, Leslie Chalmers, Keith Howker, and David Wilson as reporter.

Reality Domains

Definition

The workgroup recognized that conversations about the correspondence of a computer system to reality can move into broad discussions of human nature, the nature of perceptions of reality, etc. We

therefore, needed some meaningful and workable definitions of reality. As a result, we came to the conclusion that a helpful mechanism to examine the correspondence of an information system to reality would be to examine systems within the context of reality domains.

This concept is analogous to a security perimeter. It suggests that reality correspondence for an information system involves its correspondence to some specific domain. For example, a reality domain might be the keypunch operator. In this case, the obligation for reality testing would be to assure that the keypunch operator entered the data correctly. Broader issues such as whether or not the information was given to the keypunch operator correctly in the first place, or whether or not the transaction was properly processed, is the kind of test that does not apply to that domain. On the other hand, a demand deposit system for a bank might have a reality domain which includes all of the tellers in the bank and several other departments. In that case, we might be concerned about whether or not the demand deposit system was in fact in balance, accurate, and reflected properly the customers accounts. Thus the reality domain represents the dimension of the real world against which the system is to be tested. It was recognized that for the same system there could be several different reality domains.

Domain Content

With the general concept of domain defined, what is the content of a domain? For any chosen domain, one needs to understand the threats, vulnerabilities, assets, risks, controls, organizational structure,

organizational functions, the functions of the system that are in a reality domain.

Selecting a Domain

To select the dimensions of a domain, one should consider such issues as organizational structure, the scope of the threats and risks, the scope of the computer systems which supports one domain, the possibilities for proper implementation of segregation of duties within the domain, the assets that are controlled by the domain, and a reversibility concept. The reversibility concept suggests that for items that are being controlled within the domain, that the domain's scope must be large enough so that errors can be corrected for items that are being controlled by a domain before those detected errors leave the domain.

Related Domains

For any system, there could be more than one domain. Multiple domains could be viewed as concentric circles like peels of an onion, where for any domain there was another domain that was completely contained within that domain, or was completely contained by another domain. This model effectively explains such situations as the relationship between a bank teller, a branch office, and the whole bank when looking at a demand deposit system. On the other hand, it is quite possible that reality domains would not be concentrically contained within each other, but instead be separate individual domains. These two different relationships of domains could lead to different implementation approaches.

If the domains were concentric then the interface between domains could be fairly simple and straightforward. But if the domains were

not concentric, then there would need to be a formal hand-off of transactions as they enter and exit a domain. A simple example of this kind of hand-off is the process of delivering cash through a teller's window. First, the teller counts the cash - that is one domain; the money is then passed through the window and the person receiving the money counts it again - the second domain. There was a formal exchange process at the domain boundary. This need for a formal domain boundary process for the flow of transactions from domain to domain may drive the need for the labeling of data within an information system. It may be necessary for the application system to be able to formally recognize its own internal subcomponents in a way that ties back to reality domains.

There also needs to be formal boundary recognitions as transactions cross from domain to domain, especially when they are not concentric. Finally, there needs to be some mechanism established for dealing with error recovery of transactions that cross domains.

Domain Summary

The group worked with this domain model and applied it to a number of different examples to see if it seemed to make sense. Although the nature of this review was admittedly limited, the reality domain concept seems to hold up well as a way to look at problems and seek practical implementation approaches. It also corresponds well to the concept of security perimeter which is well understood, and other similar concepts found in audit and evaluation of systems of internal control.

Segregation of Duties - Checking the Correspondence of a Domain to Reality

Once a domain has been selected, and the appropriate threats, vulnerabilities, assets, risks, organizational structure and functions, and controls have been understood, the next step is to identify what information from the information system is needed to check the correspondence of the information system to the reality domain. The correspondence checking could involve the review of appropriate inputs to the system, review of appropriate outputs from the system, and indicators that the system processed properly. Other checks to the domain include review of the values of the data and information within the system, as well as the proper functioning of programs.

Once it is determined what data is needed from a system, the actual checking of the correspondence of an applications system to a reality domain must follow Rule C3 for suitable segregation of duties. The segregation of duties concept implies the following:

1. No individual can have such capabilities and functions to cause substantial damage unless the person is in collusion with someone else. By damage, this implies both errors as well as intentional acts.
2. Every verification process which verifies the reality domain to the application system must be a process itself which involves two or more people, working independently.

There was consensus in the workgroup that the segregation of duties principle was reasonable. If two or more people independently verify correspondence to an input transaction, review output reports, or review process results, that constitutes a reasonable process that meets generally accepted standards. There was a discussion for which there was no real closure about weak vs. strong segregation of duties. Specifically, segregation of duties which involves straight dual control where two people do exactly the same thing twice, such as key verification, is a weaker form of segregation of duties than when two people use different independent processes to come to the same result, which is felt to be a much stronger segregation of duties. The group felt that this would be an area for further research.

The Clark-Wilson Rules

The group's next discussion focused on the adequacy of the rules that apply to the correspondence of the system to reality. These rules are C1 and C3. C1 states that all IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run. C3 states that the list of relations in E2 must be certified to meet the separation of duty requirement. It was the feeling of the group that C1 should be more explicitly stated to recognize both the need for an internal integrity as well as external integrity when an IVP is run. Further, the external integrity process may actually involve some manual procedures such as the counting of inventory that need to

be defined as part of an IVP. Clearly, this is a highly application dependent kind of activity. There was a general feeling that C3 was an adequate rule as stated.

Future Research

Finally, the group turned its attention towards future research required. Several areas of research were identified:

1. More research is needed in the concept of domains. In a way that is analogous to the implementation criteria outlined in the D.O.D. Evaluation Criteria for Trusted Systems, there may well be a need for integrity labeling of data. For instance, it might be reasonable to identify the date and time that an IVP was run to check the validity of specific CDIs. It might also be appropriate to label information so that its origins could be tracked as it passes across domains. More work must also be done on mechanisms for across domain error recovery.
2. The assertions about weak and strong segregation of duties should be further explored and understood. Clearly, segregation of duties can be implemented in a right way and a wrong way, and maybe some of the right ways are better than other right ways. More definitive research needs to be done to articulate specific implementation approaches for segregation of duties.

3. A major area of research is to determine how much of the functionality in implementing the certification rules of C1 and C3 can be transferred to become enforcement rules. This means the codification of things which are currently being viewed as application dependent. There are several approaches to dynamic segregation of duties that tie back to the authorization process for individuals.

Summary

Although we recognize that the correspondence of a computer system to reality is highly dependent upon specific applications, the model seemed to provide reasonable guidance in defining a set of practices so that systems can be certified to consistently meet generally accepted practices for data integrity.

9.0 COMMENTS ON THE INTEGRITY MODEL

David D. Clark and David R. Wilson

While it is still a bit premature to provide a complete revision of the original paper, a number of observations have been made about the model which contribute substantially to the understanding and development of the ideas. We attempt to capture some of those here.

We would like to express our appreciation to those who have taken the time to read and comment on our work. Both at this workshop and elsewhere, the comments we have received have reflected a very considerable effort by workers in the field, and we want to thank those who have contributed to our better understanding of our original efforts.

Spheres of Applicability

It is clear that the distinction between military and commercial practices should not be made a central issue in the development of the model. While the proposed model of integrity arose from a study of commercial practices, both the military and the commercial world have clear and obvious need for both assurance of integrity and control of disclosure.

The development in the paper is based on the fact that the only formal model available for security is the lattice model, which arose out of military practice and is not well understood in the commercial sector. This approach to presenting the material has led some to conclude that we believed that the military was not concerned with integrity. Clearly, this is not the case. In fact, our goal was quite the reverse: by proposing an approach to data integrity, we hoped to increase the chance that a single system might prove useful in both sectors.

Terminology

A number of terms used in the paper have been a cause of confusion. To some extent, this is because several words have taken on highly specialized meaning within the field of security.

Most obvious is 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. To some people, however, the word has come to mean the specific example of that sort of mechanism which enforces the lattice model.

Another word is "certification". The paper used the word with a general meaning; a program was certified if some procedure had been used, acceptable in the context, to assure that the Transformation Procedure operates correctly. But certification has come to mean a particular procedure, developed as part of the Orange Book methodology. Perhaps some other word might better be used for this function in the paper: verification, validation or assurance. But most seem to have overly rich implications. In any case, the intent of the authors was to admit a broad range of techniques.

Perhaps the most basic word problem centers on "model". To some, a model has a degree of completeness which our model lacks. Because we must mix together application level functions with system enforcement, it was suggested that "framework" might be a better word to capture our approach to integrity.

Initial Verification Procedures

There has been much confusion as to the purpose and requirement for the IVP. Indeed, the form and function has become much clearer since the writing of the paper. In the book-keeping example, the IVP corresponds to balancing the books. It is a procedure which is run outside the normal pattern of transactions to verify the consistency of the system.

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 be initially in a valid state. The IVP was proposed to insure that.

However, one could use a very limited IVP if this were its only purpose. For example, one could postulate an initial state in which the system was empty of data, which by definition is valid, and then use TPs from then on to operate the system. (An empty system is valid because we are concerned with data integrity: if there is no data there can be no corrupted data.)

In fact, we saw the IVP as a much more practical tool. 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. In fact, this is true as a practical matter. Consider again the comparison with balancing the books. The books are balanced once a year, even though good controls have been exercised on normal activities throughout the year. We need the IVP in the model to capture this idea, accepted in practice, that a system needs a periodic cross-checking.

In fact, the need for the IVP can be seen as yet another example of separation of duty. By having two, distinct procedures to insure integrity, it is necessary to subvert both of them to permit undetected corruption. While the detection provided by the IVP may not be timely, it is none the less a deterrent.

If the IVP is to play this role as a serious cross-check, it is necessary that it not only check the internal consistency of the data, but verify the consistency between the data and external reality. The paper does not discuss this idea, and indeed it is hard to discuss it in any formal way; activities outside the computer system, such as verifying inventory, are hard to cast as a part of computer integrity model. Through separation of duty, we deal indirectly with the issue of external consistency, but when direct actions are taken externally, we can only address these peripherally.

Given this, the importance of the external component of the IVP must be understood in the larger context of integrity. Normal business practice make verification of the real state of the world a central part of an audit. If the IVP is the computer manifestation of an audit function, then the IVP must be seen as a part of larger function which verifies the consistency of the total system: computer and reality.

Most interesting, from a study of methodology, are those systems in which the internal data has no obvious external equivalent. It is difficult, for example, to talk about the integrity of predictions of the future (such as forecasts of the weather or elections) until the time has elapsed and a cross-check with reality is possible. Perhaps one of the limits of this model is that it is most effective only in those cases where an external reality is part of the context.

Separation of Duty

A major component of this model is that the correspondence between computer and real-world information is insured by having several people perform distinct parts of the computerized actions. This separation prevents one person from adjusting the computer system so that it is internally consistent but divergent from the real world. This adjustment can happen by intent or by error. Separation of duty is effective both in detecting errors and in detecting fraud.

As a normal practice, this principle is well understood, but it is very difficult to formalize. Indeed, one working group of this workshop addressed only the issue of how to better characterize the function of separation of duty. This area is probably the least well understood part of the model.

As the paper points out, the most basic rule of separation is that the person who can create a TP cannot be permitted to run it. That is, it takes two people to bring a TP into execution.

There are several points to be made about separation of duty, particularly as it relates to the control of fraud. As the above example indicates, separation rules are stated in terms of people. It is people who commit fraud, not processes or principals. If we are to use separation of duty to control fraud, then the rules must be stated in terms of people. But this raises a fundamental problem about the computer and the real world. How is a computer to tell if two principals registered on the system in fact belong to the same person? The only mechanism available is a further use of separation of duty: insure that two people are required to register a user on the system, and construct procedures to insure that all users are associated with a computer-visible and unique identification of the person.

This need to tell which actual person goes with which computer registered entity represents a further problem. In some cases it may be unacceptable or impossible to force a person to prove who they are in such a way that their identity can be assured. One is not expected to produce a birth certificate to get a computer account, and even that level of check can fail; people do succeed in getting multiple Social Security numbers or passports. In other cases it may be an invasion of privacy to ask persons to identify themselves to that extent. Especially within computer systems, there is a concern that providing the computer with a unambiguous identifier for a person would permit undesirable forms of cross-correlation of information.

There are degrees of separation. The most primitive is to have same operation performed twice, by different people. A more complex and effective version is to structure the overall tasks and arrange the parts so that the persons performing the different parts have conflicting motives. This creates a tension between them which helps prevent collusion from arising. But the computer system is not going to be capable of deducing such things as conflict of motive, so the determination that the separation is sufficient is usually going to involve human assessment; another form of certification.

That need being recognized, there is still a role for computer enforcement of separation. Essentially, what is needed (by analogy with the lattice model for disclosure) is a partition model, in which the users performing the parts of an action are verified by the system as being partitioned into disjoint sets. The partition model might well be the mandatory component of the separation of duty enforcement, with further assessments such as

conflict of motive being left as a discretionary constraint on the partitions.

Mandatory Mechanisms

A matter which received little attention in the paper is the question of what aspects of the integrity model ought to be viewed as mandatory. In the early days of developing mechanisms for disclosure control, it was not clear that having only levels and categories as mandatory controls was sufficient. Only after much discussion and evaluation was this consensus reached.

A similar uncertainty applies to the integrity case. Especially in the enforcement of separation of duty, it is not clear which facilities should be a part of the mandatory core of the system. The previous section suggested that a simple partition model might be mandatory, while additional constraints might be discretionary. This degree of functionality is suggested by a parallel to the mandatory aspect of disclosure control. But even a simple partition model would require a great deal of information in the security kernel. It would be necessary to characterize all possible sequences of TPs for which a partition check would apply. (Such sequences are not now made explicit to the system, since separation of duty is not directly checked by systems of today.)

Alternatively, separation of duty might not be a part of the mandatory enforcement at all. Tools could be made available to the system auditors to inspect the degree of separation, but no run-time checks would be made to see that the rules had been properly set.

While we would prefer to see an experiment in which the enforcement was mandatory, this discussion does show an important difference between disclosure and integrity controls. As proposed above, integrity controls depend on the system knowing the sequence of steps over which the partition test must be applied. It might be possible for the system to derive such a list dynamically, but such is not obvious. Failing that, the list will have to be supplied by a person. This means that successful enforcement depends on the setting of rather complex lists, much more complex than the setting of a simple security label on a piece of information. At a minimum we must find a way of structuring a system so that the default action if a list is improperly set is to restrict as opposed to permit an action.

Indeed, the role of people with special privileges is more prevalent in this model compared to the disclosure control environment. People must register users, validate TPs, set up separation of duty, and so on. Probably the most important further contribution to this model is to discover ways of automating these functions, so that the overall system

correctness less and less depends on the correct judgement and reliability of humans.

10.0 AN INTEGRITY POLICY: THE WORKSHOP AS A BEGINNING

Stuart W. Katzke

National Bureau of Standards

I. INTRODUCTION

This Workshop on Integrity Policy in Computer Information Systems (WIPCIS) begins the complex process of moving from the conceptual ideas proposed in the Clark and Wilson (C&W) paper to the embodiment of an integrity policy in users' applications. The process is complex because it involves numerous factors which are beyond the control of any individual or organization. It can only succeed if the resulting integrity policy meets the requirements of a broad spectrum of government and commercial sector users and its technical specifications permit unambiguous implementation.

The Working Group Reports suggest areas where additional work is needed to advance the development of an integrity policy. NBS has agreed to coordinate and support the continuation of the integrity policy development. This paper lays out a plan for continuing this effort by identifying a set of activities that should be completed to reach the goal of user implementations of an integrity policy and identifies methods that can be used to achieve this goal. The plan allows for the possibility that several policies may be developed in order to meet different types of requirements. The Appendix to this Report contains one proposed alternative. Others will be presented in the future.

II. CONTINUATION OF THE DEVELOPMENT ACTIVITY

The following steps will be taken in conjunction with the workshop summaries and recommendations contained in this report:

Post-Workshop Review by Organizing Committee. The organizing committee will meet to review the comments on the workshop report that were submitted by the workshop participants, the comments made by the participants during the closing session of the workshop, and any other sources of input to the committee.

Identification of Additional Issues. During the review meeting, the organizing committee will identify additional issues that were of concern to the workshop participants and determine which of those should receive attention.

Solicition of Papers/Views. Position papers will be solicited which propose additions, clarifications, modifications,

and alternatives to the C&W paper. The papers will form the starting point for discussion at the second WIPCIS workshop (see below). Examples of areas which might be addressed are:

o Scope of integrity problem.

It was pointed out that there are aspects of the integrity problem that the C&W paper does not address such as loss of integrity due to natural hazards, accidents or electrical malfunctions. In addition, there are several definitions and interpretations of integrity. An integrity policy must clearly identify the scope of the integrity problem it addresses.

o Applicability of the policy.

Once the scope of the policy has been specified, questions regarding its applicability arise such as: Is it general enough to represent all applications? What type of applications are best represented by the policy? Can the policy be extended (and in what way) to represent other types of applications?

o Compatibility of the policy with existing architectures.

The policy must ultimately be implemented in computer applications on existing computer architectures. Questions to be considered include: What types of architectures will best support the policy? Is the policy compatible with Orange Book systems? What operating system functionality is required to support the policy?

o Performance/Cost tradeoffs.

Implementations of an integrity policy that degrade system performance or significantly increases operational costs will not be used. Final selection of a policy must take these factors into consideration.

o Integrity Metrics.

Associated with the policy should be a metric for indicating the integrity of a system and a metric for determining the degree of trust placed in the operational integrity of a system.

Prototype Implementations. Development of an integrity policy must include plans for prototype implementations to demonstrate the feasibility, utility, and practicallity of a proposed policy. As is pointed out in the Working Group Report on Assurance, the field of computer science harbors many examples of ideas, techniques, and proposals that seemed plausable but were either not able to be implemented, inadequate, or incorrect. Prototype implementations of the C&W policy should begin soon after the Workshop Report is completed.

Conduct the Second (and possibly other follow-on) WIPCIS Workshop. The National Bureau of Standards will host the next workshop in Gaithersburg, MD. The purpose of the workshop will be to obtain closure on an integrity policy that is applicable to a significant portion of the Government and commercial sectors. While the workshop is expected to focus on modifications and enhancements to the C&W policy, it is possible that alternative approaches will be proposed. The need for additional workshops or other methods to obtain closure on one or more integrity policies will be determined after this workshop.

III. POST-DEVELOPMENT ACTIVITIES

The development effort (outlined above), must terminate with a single or several well defined integrity policies. The following set of activities are necessary to move the integrity policies from the development arena to implementations in user applications (the set of activities described below applies to each of the policies that may result from the development effort--for the remainder of this section I will refer to only a single policy for simplicity).

Standardization. As the policy nears completion and its technical specifications harden, it may be appropriate to introduce a draft standard into one or more of the voluntary standards organizations, such as ANSI or IEEE, in order to obtain a national consensus. NBS will work within the voluntary standards process to assist in achieving this goal. In addition, NBS will consider its publication as a FIPS Standard or Guideline. For the remainder of this paper, it is assumed that the policy will be approved as a national or FIPS standard.

Test Implementations of the Standard. During the standardization process (i.e., prior to acceptance as a standard), it is important to develop laboratory-based test implementations of the proposed standard. Test implementations often uncover inconsistencies, ambiguities and errors in a standard's specifications. In addition, issues such as applicability, compatibility, interoperability and performance/cost tradeoffs are more easily addressed in this environment.

Validation Criteria and Testing. Acceptance of a standard is facilitated by the availability of validation suites that test user/vendor implementations of a standard for conformance with the standard. Consequently, validation criteria and prototype implementations of validation tests should be completed no later than user/vendor implementations are available. Decisions that must be made during this activity include which organization(s) will develop the validation criteria and prototype implementations and which organization(s) will do production

testing of user/vendor implementations. NBS has had much experience in this area and offers to provide guidance and assistance to facilitate completion of this activity.

User Requirements/Vendor Products. It is anticipated that several useful formal/informal policies will be derived during this activity, including some variation of the C&W policy. Some of these may be applicable to a wider number of organizations than others. If a policy is developed that has extremely widespread applicability to many organizations and follows the standardization process (as described above), then it may be reasonable for vendors to incorporate the policy into their base operating systems. However, there are many that believe that base operating systems should not implement policies but should provide basic security capabilities to support the implementation of an organization's chosen policy. In any case, the focus of this activity should be the development of meaningful policies that "fit" some significant class of real world applications, not the incorporation of specific policies into vendors' base operating systems.

IV. METHODS

The following methods should be used to provide the environment and the resources for supporting the activities described above.

Workshops/Conferences. Workshops provide the vehicle for review of technical concepts and for generating new ideas. Conferences are necessary to communicate new concepts and ideas to those in computer security and related fields who have not participated in the development effort. Both are necessary to obtain widespread acceptance of the final integrity policy.

Publications/Presentations. Both of these are necessary to communicate technical concepts and development results to the computer security and related communities. They serve the same function as or are used in conjunction with workshops and conferences.

Laboratory Support. An integrity policy should not be proposed for widespread use until it has been implemented and thoroughly tested in a laboratory environment. Laboratory activities should be used to determine the correctness, feasibility, utility, applicability, and compatibility of proposed integrity policies. Other laboratory efforts include development of validation criteria, and conformance and interoperability testing.

User/Vendor Cooperative Projects. Cooperative projects usually involve contributions of resources by several cooperating parties to achieve common goals. Resources include people (e.g.,

guest workers), equipment, and software. The laboratory environment and its associated activities provides an excellent opportunity for cooperation among parties interested in supporting the development an integrity policy.

Research Grants. If rapid progress is to be made in the development of an integrity policy, it is nesessary to continue to support the development and post-development activities described above. To date, work that led to the C&W paper has been supported by Ernst and Whinney. Although they have been most generous in their support, the organizing committee should solicit research grants from government and private sources in order to significantly speed up the process. The organizing committee could accept grants and then redirect the funds to high priority areas or could recommend that grants be given directly to researchers in those areas.

V. CONCLUSIONS

The level of interest generated by the C&W paper indicates that an integrity policy is needed in both the Government and private sectors and the Working Group Reports conclude that the C&W model represents an excellent first approximation to such a policy. Each of the Working Groups has recommended additional areas of investigation necessary to continue the development of the policy. NBS has agreed to nurture this activity and has recommended a plan of action.

While this workshop is an excellent beginning, we must not stop until we have reached consensus on an integrity policy. Let us continue to work together to achieve this goal.

A1

Information Systems Consulting℠

A Comparison of Commercial and Military Computer Security Policies

by David D. Clark, *Senior Research Scientist*
MIT Laboratory for Computer Science
David R. Wilson, *Director, Information Security Services*
Ernst & Whinney

Ernst & Whinney

Abstract Most discussions of computer security focus on control of disclosure. In particular, the U.S. Department of Defense has developed a set of criteria for computer mechanisms to provide control of classified information. However, for that core of data processing concerned with business operation and control of assets, the primary security concern is data integrity. This paper presents a policy for data integrity based on commercial data processing practices, and compares the mechanisms needed for this policy with the mechanisms needed to enforce the lattice model for information security. We argue that a lattice model is not sufficient to characterize integrity policies, and that distinct mechanisms are needed to control disclosure and to provide integrity.

A Comparison of Commercial and Military Computer Security Policies

Introduction

Any discussion of mechanisms to enforce computer security must involve a particular security policy that specifies the security goals the system must meet and the threats it must resist. For example, the high-level security goals most often specified are that the system should prevent unauthorized disclosure or theft of information, should prevent unauthorized modification of information, and should prevent denial of service. Traditional threats that must be countered are system penetration by unauthorized persons, unauthorized actions by authorized persons, and abuse of special privileges by systems programmers and facility operators. These threats may be intentional or accidental.

Imprecise or conflicting assumptions about desired policies often confuse discussions of computer security mechanisms. In particular, in comparing commercial and military systems, a misunderstanding about the underlying policies the two are trying to enforce often leads to difficulty in understanding the motivation for certain mechanisms that have been developed and espoused by one group or the other. This paper discusses the military security policy, presents a security policy valid in many commercial situations, and then compares the two policies to reveal important differences between them.

The military security policy we are referring to is a set of policies that regulates the control of classified information within the government. This well-understood, high-level information security policy is that all classified information shall be protected from unauthorized disclosure or declassification. Mechanisms used to enforce this policy include the mandatory labeling of all documents with their classification level, and the assigning of user access categories based on the investigation (or "clearing") of all persons permitted to use this information. During the last 15 to 20 years, considerable effort has gone into determining which mechanisms should be used to enforce this policy within a computer. Mechanisms such as identification and authorization of users, generation of audit information, and association of access control labels with all information objects are well understood. This policy is defined in the Department of Defense (DoD) Trusted Computer System Evaluation Criteria, often called the "Orange Book" from the color of its cover. It articulates a standard for maintaining confidentiality of information and is, for the purposes of our paper, the "military" information security policy. The term "military" is perhaps not the most descriptive characterization of this policy; it is relevant to any situation in which access rules for sensitive material must be enforced. We use the term "military" as a concise tag that at least captures the origin of the policy.

In the commercial environment, preventing disclosure is often important, but preventing unauthorized data modification is usually paramount. In particular, for that core of commercial data processing that relates to management and accounting for assets, preventing fraud and error is the primary goal. This goal is addressed by enforcing the integrity rather than the privacy of the information. For this reason, the policy we will concern ourselves with is one that addresses integrity rather than disclosure. We will call this a commercial policy, in contrast to the military information security policy. We are not suggesting that integrity plays no role in military concerns. However, to the extent that the Orange Book is the articulation of the military information security policy, there is a clear difference of emphasis in the military and commercial worlds.

While the accounting principles that are the basis of fraud and error control are well known, there is yet no Orange Book for the commercial sector that articulates how these policies are to be implemented in the context of a computer system. This makes it difficult to answer the question of whether the mechanisms designed to enforce military information security policies also apply to enforcing commercial integrity policies. It would be very nice if the same mechanisms could meet both goals, thus enabling the commercial and military worlds to share the development costs of the necessary mechanisms. However, we will argue that two distinct classes of mechanism will be required, because some of the mechanisms needed to enforce disclosure controls and integrity controls are very different.

Therefore, the goal of this paper is to defend two conclusions. First, there is a distinct set of security policies, related to integrity rather than disclosure, which are often of highest priority in the commercial data processing environment. Second, some separate mechanisms are required for enforcement of these policies, disjoint from those of the Orange Book.

Military Security Policy

The policies associated with the management of classified information, and the mechanisms used to enforce these policies, are carefully defined and well understood within the military. However, these mechanisms are not necessarily well understood in the commercial world, which normally does not have such a complex requirement for control of unauthorized disclosure. Because the military security model provides a good starting point, we begin with a brief summary of computer security in the context of classified information control.

The top-level goal for the control of classified information is very simple: classified information must not be disclosed to unauthorized individuals. At first glance, it appears the correct mechanism to enforce this policy is a control over which individuals can read which data items. This mechanism, while certainly needed, is much too simplistic to solve the entire problem of unauthorized information release. In particular, enforcing this policy requires a mechanism to control writing of data as well as reading it. Because the control of writing data is superficially associated with ensuring integrity rather than preventing theft, and the classification policy concerns the control of theft, confusion has arisen about the fact that the military mechanism includes strong controls over who can write which data.

Informally, the line of reasoning that leads to this mechanism is as follows. To enforce this policy, the system must protect itself from the authorized user as well as the unauthorized user. There are a number of ways for the authorized user to declassify information. He can do so as a result of a mistake, as a deliberate illegal action, or because he invokes a program on his behalf that, without his knowledge, declassifies data as a malicious side effect of its execution.

This class of program, sometimes called a "Trojan Horse" program, has received much attention within the military. To understand how to control this class of problem in the computer, consider how a document can be declassified in a noncomputerized context. The simple technique involves copying the document, removing the classification labels from the document with a pair of scissors, and then making another copy that does not have the classification labels. This second copy, which physically appears to be unclassified, can then be carried past security guards who are responsible for controlling the theft of classified documents. Declassification occurs by copying.

To prevent this in a computer system, it is necessary to control the ability of an authorized user to copy a data item. In particular, once a computation has read a data item of a certain security level, the system must ensure that any data items written by that computation have a security label at least as restrictive as the label of the item previously read. It is this mandatory check of the security level of all data items whenever they are written that enforces the high level security policy.

An important component of this mechanism is that checking the security level on all reads and writes is mandatory and enforced by the system, as opposed to being at the discretion of the individual user or application. In a typical time sharing system not intended for multilevel secure operation, the individual responsible for a piece of data determines who may read or write that data. Such

discretionary controls are not sufficient to enforce the military security rules because, as suggested above, the authorized user (or programs running on his behalf) cannot be trusted to enforce the rules properly. The mandatory controls of the system constrain the individual user so that any action he takes is guaranteed to conform to the security policy. Most systems intended for military security provide traditional discretionary control in addition to the mandatory classification checking to support what is informally called "need to know." By this mechanism, it is possible for the user to further restrict the accessibility of his data, but it is not possible to increase the scope in a manner inconsistent with the classification levels.

In 1983, the U.S. Department of Defense produced the Orange Book, which attempts to organize and document mechanisms that should be found in a computer system designed to enforce the military security policies. This document stresses the importance of mandatory controls if effective enforcement of a policy is to be achieved within a system. To enforce the particular policy of the Orange Book, the mandatory controls relate to data labels and user access categories. Systems in division C have no requirement for mandatory controls, while systems in divisions A and B specifically have these mandatory maintenance and checking controls for labels and user rights. (Systems in Division A are distinguished from those in B, not by additional function, but by having been designed to permit formal verification of the security principles of the system.)

Several security systems used in the commercial environment, specifically RACF, ACF/2, and CA-TopSecret, were recently evaluated using the Orange Book criteria. The C ratings that these security packages received would indicate that they did not meet the mandatory requirements of the security model as described in the Orange Book. Yet, these packages are used commonly in industry and viewed as being rather effective in their meeting of industry requirements. This would suggest that industry views security requirements somewhat differently than the security policy described in the Orange Book. The next section of the paper begins a discussion of this industry view.

A Commercial Security Policy for Integrity

Clearly, control of confidential information is important in both the commercial and military environments. However, a major goal of commercial data processing, often the most important goal, is to ensure integrity of data to prevent fraud and errors. No user of the system, even if authorized, may be permitted to modify data items in such a way that assets or accounting records of the company are lost or corrupted. Some mechanisms in the system, such as user authentication, are an integral part of enforcing both the commercial and military policies. However, other mechanisms are very different.

The high-level mechanisms used to enforce commercial security policies related to data integrity were derived long before computer systems came into existence. Essentially, there are two mechanisms at the heart of fraud and error control: the well-formed transaction and segregation of duty among employees.

The concept of the well-formed transaction is that a user should not manipulate data arbitrarily, but only in constrained ways that preserve or ensure the integrity of the data. A very common mechanism in well-formed transactions is to record all data modifications in a log so that actions can be audited later. (Before the computer, bookkeepers were instructed to write in ink, and to make correcting entries rather than erase in case of error. In this way the books themselves, being write-only, became the log, and any evidence of erasure was indication of fraud.)

Perhaps the most formally structured example of well-formed transactions occurs in accounting systems, which model their transactions on the principles of double entry bookkeeping. Double entry bookkeeping ensures the internal consistency of the system's data items by requiring that any modification of the books comprises two parts, which account for or balance each other. For example, if a check is to be written (which implies an entry in the cash account) there must be a matching entry on the accounts payable account. If an entry is not performed properly, so that the parts do not match, this can be detected by an independent test (balancing the books). It is thus possible to detect such frauds as the simple issuing of unauthorized checks.

The second mechanism to control fraud and error, segregation of duty, attempts to ensure the external consistency of the data objects: the correspondence between the data object and the real world object it represents. Because computers do not normally have direct sensors to monitor the real world, computers cannot verify external consistency directly. Rather, the correspondence is ensured indirectly by separating all operations into several subparts and requiring that each subpart be executed by a different person. For example, the process of purchasing some item and paying for it might involve subparts: authorizing the purchase order, recording the arrival of the item, recording the arrival of the invoice, and authorizing payment. The last subpart, or step, should not be executed unless the previous three are properly done. If each step is performed by a different person, the external and internal representation should correspond unless some of these people conspire. If

one person can execute all of these steps, then a simple form of fraud is possible, in which an order is placed and payment made to a fictitious company without any actual delivery of items. In this case, the books appear to balance; the error is in the correspondence between real and recorded inventory.

Perhaps the most basic segregation of duty rule is that any person permitted to create or certify a well-formed transaction may not be permitted to execute it (at least against production data). This rule ensures that at least two people are required to cause a change in the set of well-formed transactions.

The segregation of duty method is effective except in the case of collusion among employees. For this reason, a standard auditing disclaimer is that the system is certified correct under the assumption that there has been no collusion. While this might seem a risky assumption, the method has proved very effective in practical control of fraud. Segregation of duty can be made very powerful by thoughtful application of the technique, such as random selection of the sets of people to perform some operation, so that any proposed collusion is safe only by chance. Segregation of duty is thus a fundamental principle of commercial data integrity control.

Therefore, for a computer system to be used for commercial data processing, specific mechanisms are needed to enforce these two rules. To ensure that data items are manipulated only by means of well-formed transactions, it is first necessary to ensure that a data item can be manipulated only by a specific set of programs. These programs must be inspected for proper construction, and controls must be provided on the ability to install and modify these programs, so that their continued validity is ensured. To ensure segregation of duties, each user must be permitted to use only certain sets of programs. The assignment of people to programs must again be inspected to ensure that the desired controls are actually met.

These integrity mechanisms differ in a number of important ways from the mandatory controls for military security as described in the Orange Book. First, with these integrity controls, a data item is not necessarily associated with a particular security level, but rather with a set of programs permitted to manipulate it. Second, a user is given authority not to read or write certain data items, but to execute certain programs on certain data items. The distinction between these two mechanisms is fundamental. With the Orange Book controls, a user is constrained by what data items he can read and write. If he is authorized to write a particular data item, he may do so in any way he chooses. With commercial integrity controls, the user is constrained by what programs he can execute, and the manner in which he can read or write data items is implicit in the actions of those programs. Because of segregation of duties, it will almost always be the case that a user, even though he is authorized

to write a data item, can do so only by using some of the transactions defined for that data item. Other users, with different duties, will have access to different sets of transactions related to that data.

Mandatory Commercial Controls

The concept of mandatory control is central to the mechanisms for military security, but the term is not usually applied to commercial systems. That is, commercial systems have not reflected the idea that certain functions, central to the enforcement of policy, are designed as a fundamental characteristic of the system. However, it is important to understand that the mechanisms described in the previous section, in some respects, are mandatory controls. They are mandatory in that the user of the system should not, by any sequence of operations, be able to modify the list of programs permitted to manipulate a particular data item or to modify the list of users permitted to execute a given program. If the individual user could do so, then there would be no control over the ability of an untrustworthy user to alter the system for fraudulent ends.

In the commercial integrity environment, the owner of an application and the general controls implemented by the data processing organization are responsible for ensuring that all programs are well-formed transactions. As in the military environment, there is usually a designated separate staff responsible for assuring that users can execute transactions only in such a way that the segregation of duty rule is enforced. The system ensures that the user cannot circumvent these controls. This is a mandatory rather than a discretionary control.

The two mandatory controls, military and commercial, are very different mechanisms. They do not enforce the same policy. The military mandatory control enforces the correct setting of classification levels. The commercial mandatory control enforces the rules that implement the well-formed transaction and segregation of duty model. When constructing a computer system to support these mechanisms, very different low-level tools are implemented.

An interesting example of these two sets of mechanisms can be found in the Multics operating system, marketed by Honeywell Information Systems and evaluated by the Department of Defense in Class B2 of its evaluation criteria. A certification in Division B implies that Multics has mandatory mechanisms to enforce security levels, and indeed those mechanisms were specifically implemented to make the system usable in a military multilevel secure environment [WHITMORE]. However, those mechanisms do not provide a sufficient basis for enforcing a commercial integrity model. In fact, Multics has an entirely different set of mechanisms, called protection rings, that were developed specifically for this purpose

[SCHROEDER]. Protection rings provide a means for ensuring that data bases can be manipulated only by programs authorized to use them. Multics thus has two complete sets of security mechanisms, one oriented toward the military and designed specifically for multilevel operation, and the other designed for the commercial model of integrity.

The analogy between the two forms of mandatory control is not perfect. In the integrity control model, there must be more discretion left to the administrator of the system, because the determination of what constitutes proper segregation of duty can be done only by a comparison with application-specific criteria. The segregation of duty determination can be rather complex, because the decisions for all the transactions interact. This greater discretion means that there is also greater scope for error by the security officer or system owner, and that the system is less able to prevent the security officer, as opposed to the user, from misusing the system. To the system user, however, the behavior of the two mandatory controls is similar. The rules are seen as a fundamental part of the system, and may not be circumvented, only further restricted, by any other discretionary control that exists.

Commercial Evaluation Criteria

As discussed earlier, RACF, ACF/2, and CA-TopSecret were all reviewed using the Department of Defense evaluation criteria described in the Orange Book. Under these criteria, these systems did not provide any mandatory controls. However, these systems, especially when executed in the context of a telecommunications monitor system such as CICS or IMS, constitute the closest approximation the commercial world has to the enforcement of a mandatory integrity policy. There is thus a strong need for a commercial equivalent of the military evaluation criteria to provide a means of categorizing systems that are useful for integrity control.

Extensive study is needed to develop a document with the depth of detail associated with the Department of Defense evaluation criteria. But, as a starting point, we propose the following criteria, which we compare to the fundamental computer security requirements from the "Introduction" to the Orange Book. First, the system must separately authenticate and identify every user, so that his actions can be controlled and audited. (This is similar to the Orange Book requirement for identification.) Second, the system must ensure that specified data items can be manipulated only by a restricted set of programs, and the data center controls must ensure that these programs meet the well-formed transaction rule. Third, the system must associate with each user a valid set of programs to be run, and the data center controls must ensure that these sets meet the segregation of duty rule. Fourth, the system must maintain an

auditing log that records every program executed and the name of the authorizing user. (This is superficially similar to the Orange Book requirement for accountability, but the events to be audited are quite different.)

In addition to these criteria, the military and commercial environments share two requirements. First, the computer system must contain mechanisms to ensure that the system enforces its requirements. And second, the mechanisms in the system must be protected against tampering or unauthorized change. These two requirements, which ensure that the system actually does what it asserts it does, are clearly an integral part of any security policy. These are generally referred to as the "general" or "administrative" controls in a commercial data center.

A Formal Model of Integrity

In this section, we introduce a more formal model for data integrity within computer systems, and compare our work with other efforts in this area. We use as examples the specific integrity policies associated with accounting practices, but we believe our model is applicable to a wide range of integrity policies.

To begin, we must identify and label those data items within the system to which the integrity model must be applied. We call these "Constrained Data Items," or CDIs. The particular integrity policy desired is defined by two classes of procedures: Integrity Verification Procedures, or IVPs, and Transformation Procedures, or TPs. The purpose of an IVP is to confirm that all of the CDIs in the system conform to the integrity specification at the time the IVP is executed. In the accounting example, this corresponds to the audit function, in which the books are balanced and reconciled to the external environment. The TP corresponds to our concept of the well-formed transaction. The purpose of the TPs is to change the set of CDIs from one valid state to another. In the accounting example, a TP would correspond to a double entry transaction.

To maintain the integrity of the CDIs, the system must ensure that only a TP can manipulate the CDIs. It is this constraint that motivated the term Constrained Data Item. Given this constraint, we can argue that, at any given time, the CDIs meet the integrity requirements. (We call this condition a "valid state.") We can assume that at some time in the past the system was in a valid state, because an IVP was executed to verify this. Reasoning forward from this point, we can examine the sequence of TPs that have been executed. For the first TP executed, we can assert that it left the system in a valid state as follows. By definition it will take the CDIs into a valid state if they were in a valid state before execution of the

TP. But this precondition was ensured by execution of the IVP. For each TP in turn, we can repeat this necessary step to ensure that, at any point after a sequence of TPs, the system is still valid. This proof method resembles the mathematical method of induction, and is valid provided the system ensures that only TPs can manipulate the CDIs.*

While the system can ensure that only TPs manipulate CDIs, it cannot ensure that the TP performs a well-formed transformation. The validity of a TP (or an IVP) can be determined only by certifying it with respect to a specific integrity policy. In the case of the bookkeeping example, each TP would be certified to implement transactions that lead to properly segregated double entry accounting. The certification function is usually a manual operation, although some automated aids may be available.

Integrity assurance is thus a two-part process: certification, which is done by the security officer, system owner, and system custodian with respect to an integrity policy; and enforcement, which is done by the system. Our model to this point can be summarized in the following three rules:

C1: (Certification) All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run.

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.

E1: (Enforcement) The system must maintain the list of relations specified in rule C2, and must ensure that the only manipulation of any CDI is by a TP, where the TP is operating on the CDI as specified in some relation.

*There is an additional detail the system must enforce, which is to ensure that TPs are executed serially, rather than several at once. During the mid-point of the execution of a TP, there is no requirement that the system be in a valid state. If another TP begins execution at this point, there is no assurance that the final state will be valid. To address this problem, most modern data base systems have mechanisms to ensure that TPs appear to have executed in a strictly serial fashion, even if they were actually executed concurrently for efficiency reasons.

The above rules provide the basic framework to ensure internal consistency of the CDIs. To provide a mechanism for external consistency, the segregation of duty mechanism, we need additional rules to control which persons can execute which programs on specified CDIs:

E2: The system must maintain a list of relations of the form: (UserID, TPi, (CDIa, CDIb, CDIc, . .)), 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.

C3: The list of relations in E2 must be certified to meet the segregation of duty requirement.

Formally, the relations specified for rule E2 are more powerful than those of rule E1, so E1 is unnecessary. However, for both philosophical and practical reasons, it is helpful to have both sorts of relations. Philosophically, keeping E1 and E2 separate helps to indicate that there are two basic problems to be solved: internal and external consistency. As a practical matter, the existence of both forms together permits complex relations to be expressed with shorter lists, by use of identifiers within the relations that use "wild card" characters to match classes of TPs or CDIs.

The above relation made use of UserID, an identifier for a user of the system. This implies the need for a rule to define these:

E3: The system must authenticate the identity of each user attempting to execute a TP.

Rule E3 is relevant to both commercial and military systems. However, those two classes of systems use the identity of the user to enforce very different policies. The relevant policy in the military context, as described in the Orange Book, is based on level and category of clearance, while the commercial policy is likely to be based on separation of responsibility among two or more users.

There may be other restrictions on the validity of a TP. In each case, this restriction will be manifested as a certification rule and enforcement rule. For example, if a TP is valid only during certain hours of the day, then the system must provide a trustworthy clock (an enforcement rule) and the TP must be certified to read the clock properly.

Almost all integrity enforcement systems require that all TP executions be logged to provide an audit trail. However, no special enforcement rule is needed to implement this facility; the log can be modeled as another CDI, with an associated TP that only appends to the existing CDI value. The only rule required is:

C4: All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

There is only one more critical component to this integrity model. Not all data is constrained data. In addition to CDIs, most systems contain data items not covered by the integrity policy that may be manipulated arbitrarily, subject only to discretionary controls. These Unconstrained Data Items, or UDIs, are relevant because they represent the way new information is fed into the system. For example, information typed by a user at the keyboard is a UDI; it may have been entered or modified arbitrarily. To deal with this class of data, it is necessary to recognize that certain TPs may take UDIs as input values, and may modify or create CDIs based on this information. This implies a certification rule:

C5: Any TP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI. The transformation should take the input from a UDI to a CDI, or the UDI is rejected. Typically, this is an edit program.

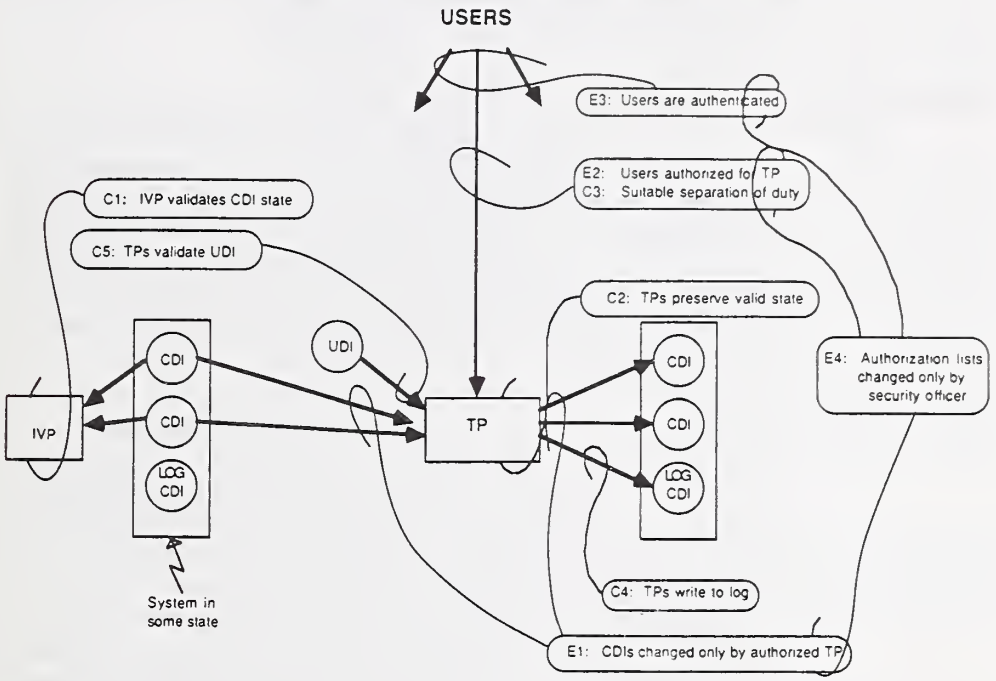
For this model to be effective, the various certification rules must not be bypassed. For example, if a user can create and run a new TP without having it certified, the system cannot meet its goals. For this reason, the system must ensure certain additional constraints. Most obviously:

E4: Only the agent permitted to certify entities may change the list of such entities associated with other entities: specifically, the list of TPs associated with a CDI and the list of users associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

This last rule makes this integrity enforcement mechanism mandatory rather than discretionary. For this structure to work overall, the ability to change permission lists must be coupled to the ability to certify, and not to some other ability, such as the ability to execute a TP. This coupling is the critical feature that ensures that the certification rules govern what actually happens when the system is run.

Together, these nine rules define a system that enforces a consistent integrity policy. The rules are summarized in Figure 1, which shows the way the rules control the system operation. The figure shows a TP that takes certain CDIs as input and produces new versions of certain CDIs as output. These two sets of CDIs represent two successive valid states of the system. The figure also shows

Figure 1: Summary of System Integrity Rules



an IVP reading the collected CDIs in the system in order to verify the CDIs' validity. Associated with each part of the system is the rule (or rules) that governs it to ensure integrity.

Central to this model is the idea that there are two classes of rules: enforcement rules and certification rules. Enforcement rules correspond to the application-independent security functions, while certification rules permit the application-specific integrity definitions to be incorporated into the model. It is desirable to minimize certification rules, because the certification process is complex, prone to error, and must be repeated after each program change. In extending this model, therefore, an important research goal must be to shift as much of the security burden as possible from certification to enforcement.

For example, a common integrity constraint is that TPs are to be executed in a certain order. In the model (and in most systems today), this idea can be captured only by storing control information in some CDI, and executing explicit program steps in each TP to test this information. The result of this style is that the desired policy is hidden within the program, rather than being stated as an explicit rule that the system can then enforce.

Other examples exist. Segregation of duty might be enforced by analysis of sets of accessible CDIs for each user. We believe that further research on specific aspects of integrity policy would lead to a new generation of tools for integrity control.

Other Models of Integrity

Other attempts to model integrity have tried to follow more closely the structure for data security defined by Bell and LaPadula [BELL], the formal basis of the military security mechanisms. Biba [BIBA] defined an integrity model that is the inverse of the Bell and LaPadula model. His model states that data items exist at different levels of integrity, and that the system should prevent lower level data from contaminating higher level data. In particular, once a program reads lower level data, the system prevents that program from writing to (and thus contaminating) higher level data.

Our model has two levels of integrity: the lower level UDIs and the higher level CDIs. CDIs would be considered higher level because they can be verified using an IVP. In Biba's model, any conversion of a UDI to a CDI could be done only by a security officer or trusted process. This restriction is clearly unrealistic; data input is the most common system function, and should not be done by a mechanism essentially outside the security model. Our model permits the security officer to certify the *method* for integrity upgrade (in our terms, those TPs that take UDIs as input values), and thus recognizes the fundamental role of the TP (i.e., trusted process) in our model. More generally, Biba's model lacks any equivalent of rule E1 (CDIs changed only by authorized TP), and thus cannot provide the specific idea of constrained data.

Another attempt to describe integrity using the Bell and LaPadula model is Lipner [LIPNER]. He recognizes that the category facility of this model can be used to distinguish the general user from the systems programmer or the security officer. Lipner also recognizes that data should be manipulated only by certified (production) programs. In attempting to express this in terms of the lattice model, he is constrained to attach lists of users to programs and data separately, rather than attaching a list of programs to a data item. His model thus has no way to express our rule E1. By combining a lattice security model with the Biba integrity model, he more closely approximates the desired model, but still cannot effectively express the idea that data may be manipulated only by specified programs (rule E1).

Our integrity model is less related to the Bell and LaPadula model than it is to the models constructed in support of security certification of systems themselves. The iterative process we use to argue that TPs preserve integrity, which starts with a known valid state and then validates incremental modifications, is also the meth-

odology often used to verify that a system, while executing, continues to meet its requirements for enforcing security. In this comparison, our CDIs would correspond to the data structures of the system, and the TPs to the system code. This comparison suggests that the certification tools developed for system security certification may be relevant for the certifications that must be performed on this model.

For example, if an Orange Book for industry were created, it also m have rating levels. Existing systems such as ACF/2, RACF, an CA-TopSecret certainly would be found wanting in comparison to the model. This model would suggest that, to receive higher ratings, these security systems must provide: better facilities for end-user authentication; segregation of duties within the security officer functions, such as the ability to segregate the person who adds and deletes users from those who write a user's rules, and restriction of the security function from user passwords; and the need to provide much better rule capabilities to govern the execution of programs and transactions.

The commercial sector would be very interested in a model that would lead to and measure these kinds of changes. Further, for the commercial world, these changes would be much more valuable than to take existing operating systems and security packages to B or A levels as defined in the Orange Book.

Conclusion

With the publication of the Orange Book, a great deal of public and governmental interest has focused on the evaluation of computer systems for security. However, it has been difficult for the commercial sector to evaluate the relevance of the Orange Book criteria, because there is no clear articulation of the goals of commercial security. This paper has attempted to identify and describe one such goal, information integrity, a goal that is central to much of commercial data processing.

In using the words commercial and military in describing these models, we do not mean to imply that the commercial world has no use for control of disclosure, or that the military is not concerned with integrity. Indeed, much data processing within the military exactly matches commercial practices. However, taking the Orange Book as the most organized articulation of military concerns, there is a clear difference in priority between the two sectors. For the core of traditional commercial data processing, preservation of integrity is the central and critical goal.

This difference in priority has impeded the introduction of Orange Book mechanisms into the commercial sector. If the Orange Book mechanisms could enforce commercial integrity policies as well as those for miiitary information control, the difference in

priority would not matter, because the same system could be used for both. Regrettably, this paper argues there is not an effective overlap between the mechanisms needed for the two. The lattice model of Bell and LaPadula cannot directly express the idea that manipulation of data must be restricted to well-formed transformations, and that segregation of duty must be based on control of subjects to these transformations.

The evaluation of RACF, ACF/2, and CA-TopSecret against the Orange Book criteria has made clear to the commercial sector that many of these criteria are not central to the security concerns in the commercial world. What is needed is a new set of criteria that would be more revealing with respect to integrity enforcement. This paper offers a first cut at such a set of criteria. We hope that we can stimulate further effort to refine and formalize an integrity model, with the eventual goal of providing better security systems and tools in the commercial sector.

There is no reason to believe that this effort would be irrelevant to military concerns. Indeed, incorporation of some form of integrity controls into the Orange Book might lead to systems that better meet the needs of both groups.

Acknowledgments The authors would like to thank Robert G. Andersen, Frank S. Smith, III, and Ralph S. Poore (Ernst & Whinney, Information Security Services) for their assistance in preparing this paper. We also thank Steve Lipner (Digital Equipment Corporation) and the referees of the paper for their very helpful comments.

References

[Bell] Bell, D. E. and L. J. LaPadula, "Secure Computer Systems," ESD-TR-73-278 (Vol I-III) (also Mitre TR-2547), Mitre Corporation, Bedford, Mass., April 1974.

[Biba] Biba, K. J., "Integrity Considerations for Secure Computer Systems," Mitre TR-3153, Mitre Corporation, Bedford, Mass., April 1977.

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

[Lipner] Lipner, S. B., "Non-Discretionary Controls for Commercial Applications." *Proceedings of the 1982 IEEE Symposium on Security and Privacy,* Oakland, Calif., April 1982.

[Schroeder] Schroeder, M. D. and J. H. Saltzer, "A Hardware Architecture for Implementing Protection Rings," Comm ACM, 3 March 1972.

[Whitmore] Whitmore, J. C. et al., "Design for Multics Security Enhancements," ESD-TR-74-176, Honeywell Information Systems, 1974.

A2 SUMMARY OF CLARK-WILSON RULES

C1: IVP Validates CDI State

All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run,

C2: TPs Preserve Valid State

All TPs must be certified to be valid. This is, they must take a CDI to a valid final state from a valid begin state. 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.

C3: Suitable Separation of Duties

The list of relations in E2 must be certified to meet the separation of duty requirement.

C4: TPs Write to Log

All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

C5: TPs Validate UDI

Any TP that takes a UDI as in input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI. The transformation should take the UDI to a CDI, or the UDI is rejected. Typically this is an edit program.

E1: CDIs Changed Only by Authorized TP

The system must maintain the list of relations specified in rule C2, and must ensure that the only (any) manipulation of any CDI is by a TP, where the TP is operating on some CDI as specified in some relation.

E2: Users Authorized to TP

The system must maintain a list of relations of the form: (User ID, TPi, (CDIa, CDIb, CDIc, ...)), 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.

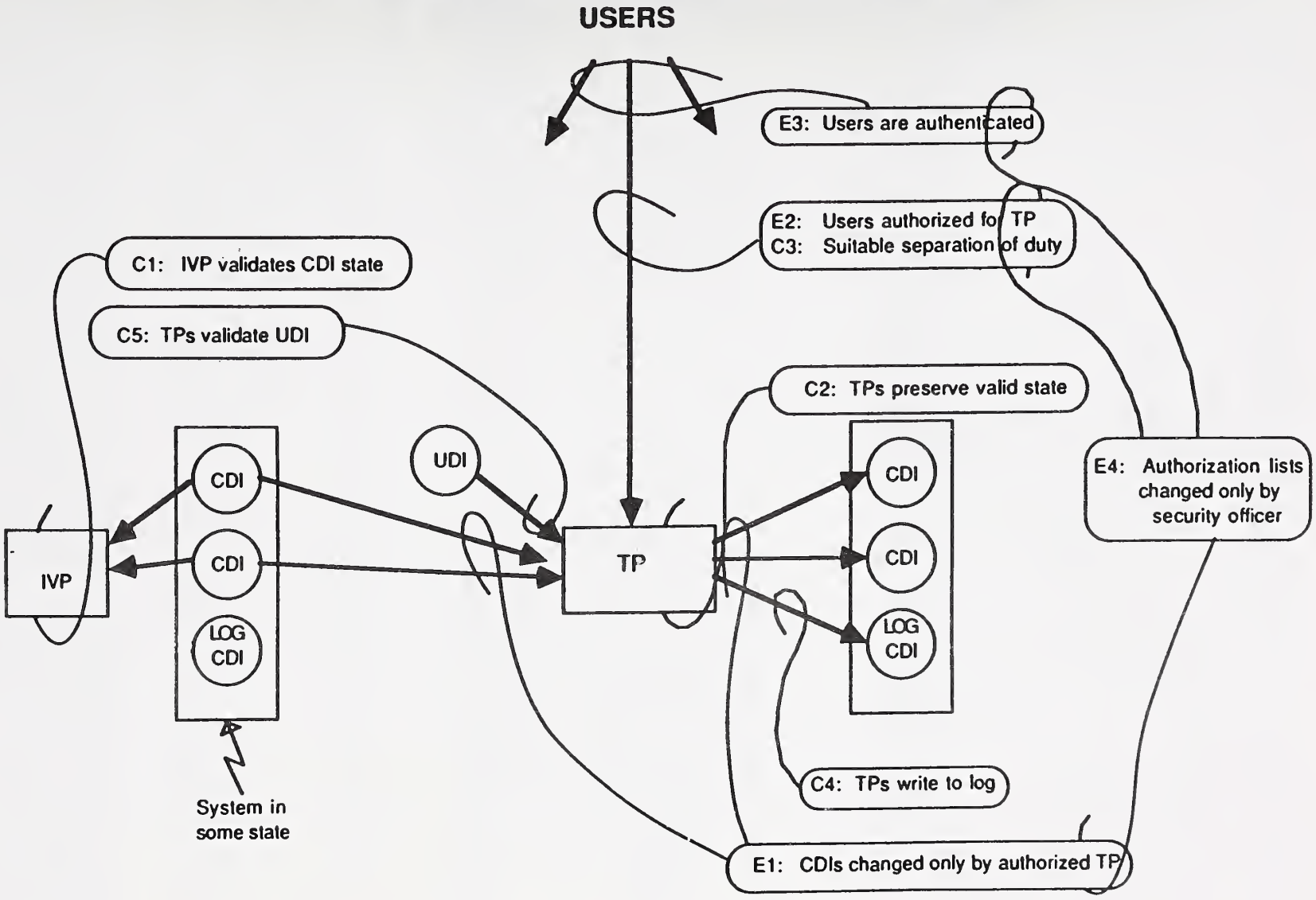
E3: Users are Authenticated

The system must authenticate the identity of each user attempting to execute a TP.

E4: Authorization Lists Changed Only by Security Officer

Only the agent permitted to certify entities may change the list of such entities associated with other entities: specifically, the list of TPs associated with a CDI and the list of users associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

Figure 1: Summary of System Integrity Rules



A3

POSITION PAPER:
WORKING GROUP ON GRANULARITY

10-19-1987

W. H. Murray
Ernst & Whinney

PREFACE

This paper is prepared as a departure point for the working group on granularity of the Workshop on Data Integrity Policy. It will be used to coordinate the activities of the various working groups and to orient the participants of the working group on granularity.

INTRODUCTION

The purpose of the Workshop is to expand on the proposals made by Clark and Wilson in their paper "A Comparison of Commercial and Military Computer Security Policies." In this paper they assert the need for a model similar to Bell-Lapadula, but aimed at the objective of data integrity rather than that of confidentiality. They propose a policy based upon the concepts of the "well formed transaction" and separation of duties.

In order to implement this policy, they propose a model consisting of nine rules. Five of these rules would be implemented external to the system and would be attested to by competent authority. The remaining four would be enforced by the computer system.

ASSUMPTIONS

(Those which are made necessary or which are implicit in Clark and Wilson.)

SOME THOUGHTS ON DATA INTEGRITY

Data can be said to have integrity when it is as good as we think it is. [Courtney] That is to say, when it agrees with our expectation.

Data can be said to have integrity when it agrees with that which it intends to describe. For example, inventory data can be said to have integrity when it tells us accurately and completely just what quantity of which items we have. Of course, it never does that, but as long as it does it well enough to support the decisions that we want to make, then it is probably good enough.

As long as the variances between the data and the inventory are not material, then they can be said to have integrity.

Data can be said to have integrity when it is worthy of our confidence. A number of things may contribute to or be necessary for such confidence. For example, we may trust data because it comes from a trusted and intended source. We trust inventory data from the inventory department and tide tables from the Oceanographic Survey.

We trust data because it is verifiable and verified. The inventory is periodically compared to the control system. If the tide tables do not agree with the observations of users, then you can expect to read about it in the New York Times.

We may trust data because it is widely published and subject to wide scrutiny. If there are errors in the tide tables, we need not expect to have to detect all of them ourselves.

We may trust data because we have evidence that it was independently verified by responsible individuals. That is why we include the auditors certification in the annual report.

An essential ingredient to our trust may be that we have evidence that the data has not been maliciously modified, for example, when all the pages are there and there are no erasures.

In early times, for most of our history, and to this day in what are called primitive or oral cultures, we achieved data integrity by disseminating widely. The more important to the society the data was thought to be, the more people we told it to. It was passed from person to person, human memory to human memory. Much of it was in poetry, ritual and song, because human memory seemed to work better on these. Tiny children memorize nursery rhymes without effort or intent.

Later, we began to record information on non-erasable, non-reusable media such as stone and clay tablets. Since this recording was expensive, it was reserved only for the most important data or for that, such as calendars, where precision and accuracy were important. Both to preserve them from generation to generation and to preserve confidentiality where indicated, such tablets have often been entrusted to a priesthood.

Before the computer, we provided for the integrity of data by recording it on paper. Paper was not quite as permanent as stone or clay but by that time we were able to make cheap carbon copies and poor photo copies. These additional copies were often created in the normal course of things and distributed to distant sites. These routine copies not only made the data available in more than one location but acted as backup copies

for each other. They also added to the integrity of the audit trail since it would be difficult to alter all of them.

The paper often had a pre-printed form or letter head which raised the cost of counterfeiting. Signatures were used to further protect against counterfeiting and to add evidence as to the identity of the originator.

Multiple signatures provided evidence of separate origination and approval, and of reconciliation and confirmation.

Documents were routinely time-stamped on receipt. Initials and rubber stamp imprints were often added as evidence of specific checks, reconciliations, control steps or verifications. These events might also be recorded in journals or logs which added to the integrity of the data and the audit trail while facilitating the location of work in process.

In the sixties, we saw the advent of shared direct access storage devices and terminals accompanied by some slight discomfort. DASD was specifically designed to be re-usable. When you changed the data on it, you did not even leave a mark.

Now, we pretended that the copy of record was still on paper and that the terminals were inquiry-only. But then some tacky person observed that if we were inquiring to the disk and making decisions based upon the data stored there, then that was the copy that counted. Some other tacky fellow observed that if you were going to make decisions based upon the data that was on the disk, then you had better use the terminal to keep it current.

At that point data security based upon media was still useful, indeed necessary, but no longer adequate. Both the problem and the term "data security" were born.

Currently, we provide for the integrity and confidentiality of data by placing it in a controlled environment. We limit access to the environment to authorized individuals. Within the environment, we limit access to the data in accordance with rules.

Since the media is re-usable, we provide for the integrity of the data by restricting the ability to update the data to only one process or user at a time. For business transaction data, that process is usually an application program or database manager. The user and author of this process are different; he who can use it cannot modify it, he who can modify it cannot use it. Thus we can have confidence that multiple people are involved and that the changes are consistent with the rules of the process.

We keep an independent record of the name of the process with the authority to update and a record of changes to that authority. We also keep a record of the event and sometimes the content of such changes. These records may be generated by the application, the database manager, or by a combination of the two.

For programs, documents or correspondence, the process is usually an individual user employing a generalized program such as an editor. While full freedom over the content is allowed to this individual user, management still has full accountability, since "write" access to the data is reserved to the individual. This restriction is imposed by the environment, usually with list-based access control.

Today we look to the system, rather than to the media, for evidence as to where or with whom the data or transaction originated.

Where the accuracy of the data is crucial, as when multiple people must rely upon it, then multiple people must be involved in its preparation. Today, we look to the system, rather than to media, to enforce this rule and to present us with the evidence that it has been effectively enforced.

Because DASD has been expensive, we have often tried to manage down the number of copies, often to only one. As soon as we are successful, we realize that we are now vulnerable to the loss of that one. We then make one or more special copies just to protect against that contingency. Of course, we usually use tape because it is cheaper and more portable.

In summary then, data can be said to have integrity if it: 1) agrees with expectation (e.g., conforms to a complete specification, is internally consistent.), 2) conforms to that which it purports to describe, 3) is worthy of our confidence, 4) originates with or is attested to by a competent source or authority, 5) verifiable and verified, 6) widely published and scrutinized, and 7) free from outside contamination or interference.

WORKSHOP

The workshop will divide into working groups.

GRANULARITY

The extent, scope, degree or effect of a control must be appropriate to the intended application. In a security context, "The test of granularity requires that the size of the resource to be controlled be small enough to constitute an acceptable risk." [Data Security Controls and Procedures, IBM Corporation,

March 1977, G320-5649]. In general, the more granular the control or the smaller the object controlled, the lower the risk. However, there is a limit: there is a point at which increased granularity increases both the level of administrative effort and the level of complexity. The first can result in inaccurate or untimely rules, while the second can mask error or malice.

In commercial applications, there are standard tests:

1. Transactions should be separately originated and approved; an individual should not be authorized to do both. In other words, not only must there be two separate transaction type programs (Clark/Wilson TPIs), but an individual should not be authorized to both.

2. The ability to create records should be separate from that to maintain them. This is the rule that requires that he who can add vendors to the payables file cannot also approve invoices for payment. Again, separate transaction types (TPIs) and no individual authorized for both.

3. The authority to change the data should be separate from the ability to set the rules under which the data should be processed. Thus, if the rules say that the data recorded by A must be checked by B, then neither A nor B can unilaterally alter the system. By implication, the user of a transaction type should not be authorized to change it, and the programmer should not be authorized to execute it. (This rule applies only to business applications and transactions. However, its application can aid in preventing the contamination of other multi-user systems.)

4. Custody of assets should be separate from access to data about the asset. For example, the cashier should not have custody of the ledger. Custodians of data should not have access to the data descriptions.

5. Other

These tests result in an accepted level of risk while keeping the administrative activity and complexity tolerable. The TPs and CDIs described in the Clark-Wilson Model likely provide the indicated granularity. However, most operating systems do not define such objects. Therefore, operating system extensions may be indicated.

A4 POSITION PAPER: ON THE USE OF MANDATORY
by W. H. Murray

TYPES OF CONTROL

The Clark and Wilson paper raises the issue of "mandatory," and that of in what sense these controls are or ought to be mandatory. In the "Orange Book," the term mandatory is used to identify a particular set of controls, i.e., those which are based upon the classification of the data and the clearance of the user rather than upon the need to know of the user. The context suggests that these controls are intended to be bound at system configuration time and to be placed thereafter beyond the control of the human managers of the system. However, the origin of the term and its intent appear to be assumed. Consequently, it is difficult to apply the term outside the context of the Orange Book and the particular set of controls described therein.

It appears to us that even this set of controls is discretionary in the sense that it is under the control of human agents. That is, a human agent assigns the data classification, enrolls the user, and grants the user credentials. Therefore, it does not appear to us that the distinguishing of controls along the current axis between "mandatory" and "discretionary" is particularly useful.

Rather, it seems to us to be more useful to distinguish controls by the time (e.g., configuration, system generation, operating system initialization, user enrollment, execution, object creation, etc.) at which they are bound, or the role of the individual (data author or owner, user manager, security staff) exercising the discretion. This set of distinctions appears to us to be more granular and descriptive than the distinction between mandatory and discretionary. If having described a particular control by the parameters that we suggest, it then appears to be useful to associate one of these labels as a shorthand notation, then so be it.

It was the experience of the builders of the AAS security subsystem that any arbitrary attempt to take something out of the hands of management and bind them within the system was self-defeating.

For example, many applications attempted to make control decisions based upon such parameters as user location type (e.g., headquarters or field) rather than explicit authority. This seemed reasonable. Since these two types were mutually exclusive, it appeared that they could be used to enforce separation of duties. However, in practice, it simply forced management into unnatural positions in an attempt to get things done. No sooner would an application build in such a rule, than,

in what seemed to be a peculiarly perverse way, the first exception to the rule would appear.

In an attempt to deal with the exceptions, management would bend or distort the system in such a manner that what was originally intended to improve control would reduce it. For example, when an individual was identified whose job required that he have two or more authorities which the application had decided should never both be held by the same user, then management would give this individual more than one user identifier and profile. While this was done in all innocence, the controls were now hopelessly confused if not compromised.

A5

POSITION PAPER:
AGREEMENT WITH THE EXTERNAL ENVIRONMENT

by W. H. Murray

It is obvious that the more closely data agrees with that which it is intended to describe, the better. Still we can work with almost any data as long as we have some idea about how bad it is. Courtney says "Data has integrity when it is no worse than we think it is."

We achieve agreement in an iterative process that begins even before the original recording of the data. The first step is to decide and describe the data that you will record. In paper systems we did this by designing a form. In early computer sytems we used a "record layout." Today we use formal data descriptions. The intent is to describe what is to be recorded, assist in proper encoding, and avoid errors of omission. By reconciling what is recorded to this description, we will gain confidence.

At recording time, we compare the new record, both to the data description and to our intent. In modern systems we do this by interactive feedback and at several levels. First, each key stroke is encoded and displayed. If the user sets out to record a "W" by striking the "W" key, then he expects to see a "W" displayed. If instead he sees a "Q" or an "E," the he knows that either he did not strike the key that he intended, or it was not properly encoded.

The system also has expectations to which it reconciles. For example, if the system says "enter customer number:" expecting a seven character numeric field followed by the "enter" key in response, then 8 characters of alphabetic characters would signal an error. The system may further expect that it has a corresponding record for the customer number entered; if it can find no record, this too may indicate an error.

"Customer number" is a coded value. The recorder has a specific customer in mind. The system can help to ensure that the number was recorded correctly by decoding it to "customer name and address" and returning that value to the user. If the values returned do not meet the recorders expectation, then either he did not encode the customer identity correctly, did not record the code correctly, it was not encoded correctly by the system, the database is in error, there is an error in programming, or perhaps even a hardware failure. In any event, corrective action is indicated in order to achieve agreement with expectation and intent.

Note that the corrective action may involve another person. Take the case in which there is no matching customer master

record for an entered number. The indicated corrective action may be the creation of an appropriate record, but the authority to do this may rest with someone else.

When an entire transaction has been recorded and each entry checked, the system may feed back a summary of the transaction for review by the recorder. Here the idea is to ensure that not only was each item recorded as intended but also that the effect is what is intended. At this time the recorder may be asked for a positive indication that the summary correspond to his intent. A record of this response can serve as a deterrent to the recorder entering data that he knows in advance to be erroneous or fraudulent.

The process does not end here. A summary of the transaction may be printed out. This step may be automatic. It is not under the control of the recorder. He does not cause it, and cannot prevent it. Instead it is prepared, from his point of view, automatically; that is, under the control of others. Copies of these may go to his management, but certainly one goes to the customer who reconciles it to his intent.

The process continues through shipment, receipt, billing, statements, remittances, deposits to banks and so on. At each step additional people compare both the data and its effect to their expectation. For example, the picker/shipper expects to find stock for an order. If there is no stock, there is either an error in the data or missing stock.

While every step offers an opportunity to detect errors, they do not offer equal opportunity to correct. The later in the process that the error is detected, the more that it will cost to correct it. For example, the National Bureau of Standards tells us that while a single key-stroke error may be corrected for a dime if detected immediately, the cost may go to two dollars if it is not detected until after the database is updated. Of course, if goods are manufactured or shipped based upon the error, the cost of correction soars.

Of course, the process can, and sometimes does, fail. People fail to reconcile or detect variances, they may dupe or collude with others, or they may fail to report or correct. The system can be made more robust by adding steps or people but only to a point. At some point, adding additional people or steps serves only to add complexity. The complexity may introduce errors of its own, or mask errors from other sources.

The design of the information system can contribute to maintaining agreement between the data and its environment. For example, application provisions for feedback and reconciliation help, as do those that provide for the involvement of multiple people. Access control mechanisms that limit access to the

application and its parts to authorized people also help. System extensions that insure that the authorizations conform to good practice are what is at issue here.

Historically, we have used a number of tests to insure that a sufficient number of people or steps are involved in business applications. These include:

1. Transactions should be separately originated and approved; an individual should not be authorized to do both. In other words, not only must there be two separate transaction type programs (Clark/Wilson TPIs), but an individual should not be authorized to both.

2. The ability to create records should be separate from that to maintain them. This is the rule that requires that he who can add vendors to the payables file cannot also approve invoices for payment. Again, separate transaction types (TPIs) and no individual authorized for both.

3. The authority to change the data should be separate from the ability to set the rules under which the data should be processed. Thus, if the rules say that the data recorded by A must be checked by B, then neither A nor B can unilaterally alter the system. By implication, the user of a transaction type should not be authorized to change it, and the programmer should not be authorized to execute it. (This rule applies only to business applications and transactions. However, its application can aid in preventing the contamination of other multi-user systems.)

4. Custody of assets should be separate from access to data about the asset. For example, the cashier should not have custody of the ledger. Custodians of data should not have access to the data descriptions.

5. Other

Systems' access control has been used to control what transactions a user could perform. However, most do not attempt to ensure that no combination of authorizations violate rules 1 and 2.

Some capability based systems, such as the IBM System/38, identify the author of a TP and maintain the identity of the TP from development to use. Thus, they have the necessary primitives and information for the application of the third rule. However, there is no enforcement. The System/38 also rigorously maintains the association between data and its description, but again makes no provision for segregation

between who changes the two. The capability-based systems offer the indicated granularity of control without a big performance, complexity or administrative hits.

A6 Data Integrity in a Business Data Processing System

01-21-1988

W. H. Murray
Ernst & Whinney

ABSTRACT

Clark and Wilson argue for a "mandatory policy" for data integrity. They would base this policy on the "well-formed transaction," and separation of duties. They suggest that there are two kinds of rules necessary to implement such a policy. One set labelled "E" are those that would be enforced by the system. The second, labelled "C" would be certified or attested to by appropriate authority. Included in this second set are the rules regarding separation of duties. They argue that these rules are too application dependent to be described with sufficient brevity and rigor to be readily enforced by the system.

This paper describes a system, IBM's Advanced Administrative System, which relies upon and implements many of the concepts suggested by Clark and Wilson. For example, it is composed exclusively of such "well-formed transactions." Additionally, it implements a function to enforce separation of duties. This function, called the "Conflict Matrix," is implemented in installation specific code. It is described here because the concepts have broader implications.

The paper gives a brief description of AAS, and its security system and shows that it implements the rules of the Clark-Wilson model. It elaborates on the Conflict Matrix function giving the history and rationale behind it, and describing the function itself. It then goes on to describe some of the general implications of this function. One implication of the history and rationale is that at least part of the rules regarding separation of duties can, and must, be system enforced.

THE ADVANCED ADMINISTRATIVE SYSTEM

The AAS was begun in 1966 and was described by Wimbrow in the IBM Systems Journal. It is a business transaction application system. Today it supports tens of thousands of users, primarily in the marketing and sales functions of IBM. It is used to record and process business transactions most of which originate in branch offices. It processes hundreds of thousands of transactions per day.

APPLICATION ORIENTATION

AAS is an application-only production-only system. That is, it does not provide any development or testing functions. These functions are both done in separate systems. Thus, a user of AAS can alter data, but cannot alter programs, procedures, or system parameters. Programmers can alter programs but only in a separate system. They can test only against test data and only in a separate system.

(This is a very strong implementation of one of the segregations of functions referenced by Clark and Wilson, i.e, that between execution and programming. While it is ad hoc, it is, nonetheless, by policy and design rather than by accident. It is supplemented by controls, implemented at user enrollment time, which prevent programmers from being enrolled as users.)

TERMINAL MONITOR PROGRAM

The terminal is owned by a terminal monitor program and not by the operating system. Therefore, there is no way in which a user of a terminal can ever see the operating system or its service calls (SVCs). There is no way in which an application, terminal monitor program or operating system failure can expose the operating system to a user.

DATABASE MANAGER PROGRAM

All application data in the system is under control of the database manager program. Application programs have access only to database services and only the database manager has access to the operating system I/O services. While the database is described to the field level and while field level services are available on an exception basis, almost all access is at the record or record segment level. These record and segment types, the entire database, are "Constrained Data Items" in the sense described by Clark and Wilson.

All access by a transaction type to record or segment type must be explicitly authorized. For reasons of brevity and performance, wild cards are allowed. We assert that this conforms to rule "E1" in the Clark and Wilson model. Again for reasons of brevity and performance, the actual table is inverted (CDI, (TPa, TPb, TPc,...)) from the form (TPi, (CDIa, CDIb, CDIc,...)) suggested by Clark and Wilson.

PROGRAM LIBRARY

Programs are stored separately from data within the AAS environment (though they look similar to the operating system). It is not possible to alter a program within the AAS environment. There are no functions available, and there is no way to

establish addressability to a program. All programs are re-entrant, and therefore, a program need not be able to address itself. Since it need not, it cannot.

Neither is it possible to execute data. Programs and data are not interchangeably addressable. The loader can access the program library and establish addressability to programs. It cannot access the database. The database manager can access data and establish addressability to it. It has no knowledge of the program library or of programs in storage.

ROLE OF TERMINAL MONITOR AND DATABASE MANAGER PROGRAMS

The AAS Terminal Monitor Program and the Database Manager Program are installation written. However, for our purposes they are part of the "system." They run as privileged extensions of the operating system where they are protected from any interference from application programs. They are written, maintained and operated separate from all applications and from each other. Therefore, we assert that all rules which they enforce can be said to be "system" enforced.

The operating system currently in use is MVS, though others have been used in the past. It should be noted that neither users nor terminals are known to MVS, but only to sub-systems such as TSO, IMS, CICS, or in this case, the AAS terminal monitor system.

AAS runs in its own MVS memory space, often as the only application in the system. Thus, even if MVS would permit it, there would be no other application in the system to interfere with it. However, when and if there are other applications running in the same system with AAS, it is protected from them by the fact that they do not have addressability to any of its resources. While it might be possible for a highly and properly privileged TSO user to HALT AAS, he could not interfere with its program or data contents.

This implementation is so foreign to many of the authors on security, that they are barely able to conceive of its implications. Many of them have only worked with systems in which the terminal is owned by the operating system and where the normal corrective action for an application failure is to return control of the terminal to the operating system, thus exposing the operating system to an application only user. They are simply unable to conceive of a system in which users are never direct or even potential users of the operating system.

TRANSACTION PROCESSING PROGRAMS

The capabilities of the system are presented to the user in the form of applications, functions, activities and transaction

types. Transactions are the most granular capabilities. Functions and activities are simply conveniently named sets of transactions within an application.

Transaction types are named, granular, limited and predictable functions. While reuseable procedures or programs may be employed across transaction types, the set of programs that will be employed by a transaction type, is fully defined by its name. The capability of a transaction type is related to a specific business function or transaction and predictable from the fully qualified name of the transaction.

Transaction types have three unique and equivalent names. The business function is described by short name which is qualified by the activity, function and application of which it is a part. For example, the CASH MAINLINE transaction might be part of the CASH activity in the PROCESS PAYMENTS function in the ACCOUNTS RECEIVABLE application.

The transaction type also has an eight character mnemonic name which may be employed by the user to refer to it. It also has a four character name that is employed by the developers and the operating environment.

After more than two decades of operation, the system has few users who can remember the business functions as they were done before the system. Therefore, most of the current users think of the business function by the name of its AAS transaction type rather than thinking of the transaction type by the name of the business function that it performs.

All manual or user input to the system are by means of these transactions. The transaction is a closed unit of work. Inputs are highly structured and limited. All inputs are edited at the field level to insure that they meet all of the requirements of the application. Inputs which do not meet the expectations of the transaction are returned to the user for correction with an "unable to process, (reason)." Only when all of the inputs to the transaction meet expectations, are any of them processed. Either the whole transaction is processed or none of it is. (Clark-Wilson rule C5: TPs validate UDI). We assert that these transaction programs are "Transformation Procedures" of the type described by Clark and Wilson.

OTHER PROGRAMS

A basic design assumption of the system and its applications is that "the real world runs in real time." Therefore, if a function can be performed via a real-time transaction, then it ought to be. Nonetheless, summarization, consolidation, report generation, document generation (e.g. checks and invoices), database reorganization, and backup and recovery functions are

performed in batch (database manager active) or offline (database manager not active) mode. All of these procedures are designed, tested and certified to leave the system in a valid state. They are "Integrity Verification Procedures" or "Transformation Procedures" of the kind described by Clark and Wilson.

THE SECURITY SUB-SYSTEM

OVERVIEW

The AAS Security System provides the mechanism for management to administer access to the system. It includes: 1) the code which requires users to identify and authenticate themselves (Clark and Wilson rule E3); 2) that which controls their access to transaction-types (Clark and Wilson rule E2); 3) by which management enrolls users and grants access to the transaction types (Clark and Wilson rule E4; 4) and that which implements a number of security related logs and alarms.

POLICY

The basic policy to be enforced by the system is that users must be limited to those capabilities that they must have to perform their assigned task. Since that task assignment comes from their manager, then it is that manager that administers the rules. The manager's authority comes from his own manager or from a designated surrogate.

However, each application has an "owner." It is this owner that specifies the controls for the application. Some of those controls are implemented by the developers in the structure and content of the transaction types. Some are implemented by the administrative rules that the owner specifies for the use of the transactions.

USER PROFILE

There is a profile in the system for each user. It is created at enrollment time. It contains the user's employee serial number, organization, owner's (the manager who enrolled the user or who is currently responsilbe for him) employee serial number, date of last change, and authority.

The authority is expressed in terms of transaction types and rules. The rules for a transaction type describe what authority the user has for it. The basic rule is "execute." However, others include "trained," and "delegate." The rules are exclusive and non-hierarchichal, i.e., no rule implies or includes another. Thus, a manager may have the authority to delegate a transaction (approve) but not the authority to execute (originate) it.

TRANSACTION SCOPE

The system provides the application developer with macros that can be used to limit the scope of a transaction to a particular organization. The organizational data used is that in the user's profile. For example, the transaction for entering orders can be limited to the user's branch office. To provide flexibility, a list of organizations for the user can be associated with the application or any part thereof.

USER IDENTIFICATION AND AUTHENTICATION

The system presents an otherwise blank screen labelled "Sign-On." The user is expected to know how to respond to this screen. No "help" is available. The expected response is the user's 6-digit employee serial number (user-chosen mnemonic names are used on most of IBM's systems) and, optionally, the six character password and the transaction-type mnemonic name. The password is generated by the system using a random number generator. It must be changed at least monthly by the user employing a "password change" transaction-type. If the password was not entered in response to the "Sign-on", the user is prompted for it on the next screen. If the transaction type was not entered, the user is presented with a menu tailored to his authority.

Until recently, users were required to enter their IDs each time they walked up to a terminal and their passwords once for each new transaction. This provided for positive identification in an environment in which terminals were shared across users but dedicated to a system. No logoff was indicated or required since a transaction was limited in scope and terminated at its normal end.

However, today it is usual for a terminal to be dedicated to a user even though they may be shared across systems. Therefore, the system is being modified to require a logoff, but not require the repeated entry of the password.

ACCESS CONTROL

When the user indicates the transaction type, the system checks to see that he is authorized to execute it. Optionally, by transaction type, it checks to insure that he is trained, i.e., has completed the system administered training required for the transaction.

ENROLLMENT AND DELEGATION

Enrollment is de-centralized. A user is enrolled by his manager. Certain controls are implemented in the enrollment process. For example, the user's default organization is set

equal to that of the manager enrolling him. The user's "owner" field is set equal to the user identifer of the enrolling manager.

The user's employee serial number is checked against the Security Profile File to insure that there is not already a profile for that user. The user's employee serial number is also checked against the Payroll Master File. The purpose of this control is to insure that only valid, current employees are enrolled. This control is included in the content of the enrollment transaction. The data required for this control is owned by the payroll application. It is provided to the Security application by the payroll application in the form of a service macro. At this time, as noted above, the system also compares the position code in the payroll record of the user against a short list of position codes associated with programmers.

Usually the enrolling manager also enters the initial profile. He can do this by selecting from a menu of those transactions that he is authorized to delegate (created from his own profile.) He can specify by transaction type mnemonic name (checked against his profile) or he can specify the name of a model profile (adjusted by his profile).

For those transaction types that check the content of the transaction against the organization of the user, the manager may expand the authority of the user beyond the user's default, but only up to the limits of his own.

The user's position code in the Payroll Master File is checked against a short list of position codes known to include development programmers. These users cannot be added with this transaction. Enrollment of programmers can only be done by the central security administrator upon written authorization of the director of information systems.

Additonal checks are made against the "conflict matrix" which is described in more detail below.

LOGS

Two logs are kept by the security sub-system. The first contains a record of any variances from security rules detected at logon or new transaction time. These include logon failures or attempts to perform unauthorized transactions. The second contains a record of all changes to user profiles. Since this record contains a reference to the previous change, it is possible to determine what the profile ever looked like and who is accountable for any authority in it.

ALARMS

Any event that results in an entry in the security variance log, also generates an alarm count. When the count exceeds a threshold, corrective action is initiated. In the early days of the system, when it owned all of the lines, the corrective action was to disable the terminal and send a message to it about the variance. Since management action was required to enable the terminal again, this was a powerful mechanism for informing management of the alarm condition. Today, the system employs a network shared with other systems and applications. Though it can still disable the port, it cannot really disable the terminal. Therefore the threshold is now set to ten rather than the old two. While no longer as effective in notifying management of the variance, the control is still effective in limiting exhaustive attacks. Today, corrective action relies more heavily upon the information in the variance journal.

THE CONFLICT MATRIX

HISTORY AND RATIONALE

After one early audit, the auditors complained of the number of instances in which users appeared to have authority that was not consistent with good practice. However, these concerns were not expressed to the owners of applications or the managers of the users, but rather to the developers.

As might be expected, the developers responded that this was a user management, rather than a technical, problem. They said that it was really not within their discretion to fix; that the concern should be taken up with the managers of the users; that the developers could not usurp the authority of these managers to determine what the users should be able to do.

At the time of this early audit, user management could still deal with the problem in a reasonable manner. However, as the number of transaction types and potential conflicts grew, it became apparent that no one could reasonably expect that a large percentage of the managers could be aware of all of these conflicts. It seemed that they would have to be given guidance. The appropriate time to give this guidance seemed to be at the time that the manager was about to create a conflict; that is, at delegation or profile maintenance time.

IMPLEMENTATION

A table of conflicts was created in a form that looked something like a user profile. The table argument was a transaction identifier and the function was a list containing the identifiers of conflicting transactions. As suggested above, the original intent was to make this information advisory only.

However, subsequent audits suggested that the number of exceptions was too high; consequently the system was changed to prohibit conflicts.

The owners of the applications suggested that the right to violate the rule should be reserved to them. Thus, the ability to create a conflict is reserved to the system or application security administrators. It may not be created by lower level administrators.

The appropriate content of the list is determined by asking the transaction-type owners to specify those transactions that are not compatible with theirs. This corresponds to Clark and Wilson rule C3 (if, and only if, no transaction type violates the rule). Since conflicts are reciprocal, each conflict generates two table entries. If the two transactions are owned by different people, then there are two separate managers, each with an opportunity to identify the conflict.

The rule is of the form IF tp1 THEN NOT tp2, tp3,... However, it is a general, system-wide, rule and need not be repeated in each user profile. (Indeed, exceptions to it are permitted in individual profiles.)

The rule is imposed and the results are bound at profile maintenance time. Thus, the cost of checking is kept low. This means that, at least in theory, additions to the conflict matrix after the creation of the profile, would not be reflected in the profile. However, such additions occur so rarely for a pre-existing pair of transactions that it is possible to check the new conflict against all of the pre-existing profiles.

We believe that this mechanism represents "enforcement" of the decision "certified" in rule C3. Further, we believe that this enforcement is essential. In any complex system, and in most significant (not to say "non-trivial") systems, it is not possible for a sufficient number of managers to know C3 for it to be adequately enforced.

Clark and Wilson might argue that this rule should be administered by the security staff, because it is mandatory, and that it is possible for one or two people to have sufficient knowledge. We would grant this but counter that only in a trivial system would it be possible for one or two people to have all of the other knowledge (e.g., job assignments) or be able to process all of the activity.

CONCLUSION

The AAS system is a system of the type described by Clark and Wilson. It fully implements all of their rules. Since it

existed prior to their model but is an example of it, it tends to validate the model.

However, we think that the experience with this system demonstrates that Clark and Wilson's rule C3 not only can, but must, be enforced by the system.

C1: IVP VALIDATES CDI STATE

All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run.

Compliance with this rule is required by the AAS programming standards, and described in the "batch program" specification. It is enforced at "promotion" time when the developer, owner and maintainer certify that the transaction type conforms to the specification.

C2: TPS PRESERVE VALID STATE

All TPs must be certified to be valid. That is, they must take a CDI to a valid final state. 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.

Compliance with this rule is required by the AAS programming standards, and described in the transaction type specification. It is enforced at "promotion" time when the developer, owner and maintainer certify that the transaction type conforms to the specification. While the list is inverted from the form shown here, it is included in the system and is used by the Database Manager Process to insure that all accesses are in compliance with this rule.

However, the individual who maintains the table is not called a "security officer," and is different from the person who performs some of the other duties designated for the "security officer." It is the responsibility of this individual to insure that access by the TP is authorized by the owner or manager of the data and is concurred in by other responsible managers. This further enhances the segregation of duties. It also reduces to a manageable level the amount of activity that any individual must process and the amount of knowledge that the indivudal must have. Therefore, for all further rules, we interpret the term "security officer" to mean the properly designated management surrogate.

C3: Suitable Separation of Duties

The list of relations in E2 must be certified to meet the separation of duty requirement.

This rule is described in the AAS system when the application owners identify transactions that must be separated from theirs to enforce appropriate segregation of function. It is bound at conflict matrix definition time and enforced at transaction delegation time.

C4: TPs Write to Log

All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstruced.

The AAS programming standards require that a TP log sufficient data to make both the event and content of its use obvious, to enable the transaction to be reversed, and to fix accountability. A protected logging service is provided to facilitate this. Compliance with this rule is described in the transaction type specification. It is enforced at "promotion" time when the developer, owner and maintainer certify that the transaction type conforms to the specification.

However, in part to protect against TP failure and in part for other reasons, there are two other separate journals kept. The AAS Security Subsystem logs the events of transaction starts and completions with reference to the user. The Database Manager Subsystem logs both the event and content of all changes to the database cross referenced to the one kept by the Security Subsystem by the transaction sequence number. Thus, the system is not wholly dependent upon the proper behavior of the TP. Both the user and the developer can be held accountable for thier actions and corrective action can be taken, independent of the log kept by the TP.

C5: TPs Validate UDI

Any TP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformation, for any possible value of the UDI. The transformation should take the UDI to a CDI, or the UDI is rejected. Typically this is an edit program.

AAS programming standards require that TPs validate all inputs. This is accomplished in three ways. First the input data is compared to expectations that are designed into the TP. These will include such things as formats (e.g., field length and character set), content (e.g., valid account numbers cannot

replicate existing ones, and old account numbers must already be reflected).

The second major way that the UDIs are validated is by "conversational feedback." In this technique coded input is decoded and fed back to the user for comparison to the user's expectation. For example, in response to customer number, the system might feedback customer name and address and ask the user for a positive acknowledgment that it matched that of the customer whose number he intended to enter.

Finally, the system employs "transaction confirmation." At the end of the TP, the TP summarizes the effect of the transaction, feeds it back to the user in the form of a summary screen, and requests one last positive acknowledgment that the effect agrees with the user's intent. Only then does the TP commit the changes to the database. The Security Subsystem then regains control, reminds the user of his accountability, and provides the transaction sequence number for recording on any supporting documentation.

Compliance with this rule is described in the transaction type specification. It is enforced at "promotion" time when the developer, owner and maintainer certify that the tranaction type conforms to the specification.

E1: CDIs Changed Only By Authorized TP

The system must maintain the list of relations specified in rule C2, and must ensure that the only (any) manipulation of any CDI is by a TP, where the TP is operating on some CDI as specified in some relation.

It is not completely clear what is intended by "the system must maintain." However, the Database Manager Process has for each CDI a list of those TPs which are authorized to update, or even access that CDI (see rule C2 above).

E2: Users Authorized to TP

The system must maintain a list of relations of the form: (User ID, TPi, (CDIa, CDIb, CDIc,...)), 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.

The AAS Security Subsystem maintains a list by user of the TPs that the user is authorized to execute. This list is separate from that which describes the CDIs that the transaction type can access. However, taken together, these lists meet the rule, if and only if, the CDIs that the TP can access are the same for all users. This is generally true by design and intent,

but with an exception. As noted above, some transactions are designed to provide more granular control by limiting a user to some subset of the named CDIs based upon his organizational unit. The organization data that the transaction will rely upon to enforce these rules is maintained and stored in the user's profile. However, since such TPs are not themselves authorized to that CDI called the "user profile" they must access this data using a service (macro) provided for that purpose by the Security Subsystem.

E3: Users Are Authenticated

The system must authenticate the identity of each user attempting to execute a TP.

User authentication, based upon a reuseable password, is performed by the AAS Security Subsystem at "logon" time.

E4: Authorization Lists Changed Only by Security Officer

Only the agent permitted to certify entities may change the list of such entities associated with other entities: specifically, the list of TPs associated with a CDI and the list of users associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

AAS restricts the ability to update any and all control tables to those specifically designed by management to do so. By design and intent, a given individual will be severely restricted as to how much such data he can effect. Most such data can be updated in real-time using appropriate TPs.

The second part of this rule appears to require a restriction over both authorizers and developers. In AAS, these restrictions are enforced, in part, by restrictions designed into the authorization TPs, and in part by ensuring that 1) developers cannot be users of the system; 2) that the authority for exceptions to rule 1 is reserved to a high level of management; 3) that even if a developer is granted an exception to rule 1 under rule 2, he still may not be authorized to execute a program for which he was responsible; and 4) there are no exceptions to rule 3.

A7

USING MANDATORY INTEGRITY TO ENFORCE "COMMERCIAL" SECURITY

Theodore M.P. Lee
Trusted Information Systems, Inc.

24 November 1987

Submitted for the
1988 IEEE Symposium on Security and Privacy

Abstract

Government research, development, and standardization efforts in computer security have been repeatedly criticized as not being applicable to the commercial world. In particular, they have been criticized as not being able to support the kinds of security policies, such as separation of duties and well-formed transactions, used by the financial and other communities to control unauthorized changes to or falsification of information. This paper demonstrates how two natural extensions -- integrity categories and partially trusted subjects -- of the principles of current DoD computer security standards could be used to implement such commercial security policies in a way that exploits the fundamental strengths of existing or future trusted systems.

1. Background

It has been alleged [Courtney], [Murray], [Clark] that the Principles of the DoD TCSEC [DoD85a] (the "Orange Book") do not apply well to the kind of security needed outside the national security establishment (NSE). In particular, it has been repeatedly stated that the TCSEC deals almost entirely with controlling unauthorized dissemination or release of information whereas in most parts of the commercial world, including the financial sector, the concern is more over controlling unauthorized modification of information. It is true that the emphasis of the TCSEC is with controlling unauthorized disclosure, since that is the primary issue that gave rise to the need for trusted computer systems in the NSE[1]. This paper will try to demonstrate, however, that the principles behind the TCSEC (mandatory access control based on labels, individual accountability, level of assurance, etc.) when properly applied

---

[1]Note, however, that one of the fundamental principles of a trusted computer system is that its kernel be self-protecting, attention to unauthorized modification is in fact inherent in the TCSEC.

do appear to give precisely the kind of controls needed in the commercial world, and in a manner that permits a systematic examination of their adequacy.

2. Fundamental Principles

Two concepts, not explicitly stated in the TCSEC but logical extensions of it, are all that is required of a system for it to enforce the kind of "commercial security policy" described by [Clark]:

1) The use of mandatory integrity categories [Biba] to control unauthorized modification of information, and

2) The use of (partially) trusted subjects, based on a variation of the revised Bell & La Padula model [Bell 86], to represent (and control) authorized transactions.

The reader is assumed to be generally familiar with the principle of associating sets of integrity categories with subjects and objects in the Biba model and in particular that integrity categories and levels have no intrinsic relation to any security categories and levels that might also be present. The concept of the (partially) trusted subject is however a relatively recent refinement of both the original Bell & La Padula model [Bell 75] and the Biba model.

A trusted subject, S, has two integrity labels, view-minimum and alter-maximum, which will be denoted here as v-min(S) and a-max(S)[2]. (An integrity label is simply a set of categories; the notion of a hierarchical integrity level is not useful in this context.) This formulation of the mandatory integrity policy model reduces to the Biba formulation merely by noting that any subject S with v-min(S) = a-max(S) is an untrusted subject. For the purpose of this paper it is assumed that mandatory control over unauthorized disclosure is of no interest so no mandatory security (confidentiality) labels or rules are included in the discussion; they could be easily added.

Given an object O, whose (single) integrity label is denoted as label(O), also a set of integrity categories, the access rules are:

(Simple Integrity Condition): S may write O if label(O) is subset of a-max(S).

(Integrity *-property): S may read O if v-min(S) is subset of label(O) or a-max(S) is subset of label(O)3.

---

[2]Note that these are the integrity duals of the v-max and a-min of [Bell86].

The first rule could also be written, using more conventional TCSEC vocabulary, as "S may write O if a-max(S) dominates label(O)", and the second rule could be written as "S may read O if label(O) dominates either v-min(S) or a-max(S)".

The security policy defined above can be described as a "Biba integrity model with partially trusted subjects". A system which implements it could, if built with the proper assurances and documentation, be evaluated against the TCSEC, especially if the Trusted Network Interpretation (TNI) [DoD87] is used as the basis, since that has several explicit discussions of integrity policies and trusted subjects.

3. Informal Interpretation

The meaning of integrity categories and the above access rules can be described in the following informal manner.

An integrity category can be interpreted as the name of a particular type of protected data where a given user (or program acting on his behalf) cannot create or modify any given type of data without explicit authorization for that type. If a given data container has data of more than one type it is labelled with a label for each type and any user or program that wishes to create or modify data in that container must be authorized for each type in it.

A set of integrity categories, as applied to an untrusted subject, such as a user, indicates that subject is authorized to create or modify data of any type in that set, but no others.

An untrusted subject can be thought of as one that cannot "create" data of a type that it hasn't already seen: any data it reads must have data of all the types it is going to write.

A (partially) trusted subject, on the other hand, can be informally said to be allowed to transform data from a limited set of input types to a limited set of output types: it can read data (or have data supplied to it) that is marked with only the set of categories specified in the view-minimum label of the subject (i.e., the data comes from at least a certain set of types) and still be allowed to write into data containers with any or all the types in its alter-maximum label. (It can also, of course, read any container that includes all the types in

---

[3]The second clause, "a-max(S) is subset of label(O)", is necessary to allow a trusted subject to view data objects it is permitted to modify, which in general would be appropriate and necessary but could not happen otherwise unless v-min dominated a-max. See paragraph (c) of section (8) below for an alternative forumulation that might be more satisfying.

the alter-maximum label too, but, as noted, this needed to be included explicitly in the model.)

4. Application to the Clark & Wilson "Commercial" Security Policy

Although it is expressed in terms of nine separate rules, which will be examined briefly in the next section, the essence of them stated roughly is:

All data (of interest) must be modified by, and only by, authorized well-formed transactions, where the principle of separation of duties is used to limit who can perform what transactions and make what changes to the system.

A B1 or higher class trusted computer system which supports the "Biba integrity model with partially trusted subjects," as its security policy can be configured and operated to enforce the Clark & Wilson policy, in an entirely natural way, by following eight administrative steps or procedures:

P1. Identify each type (role) of non-administrative (operational) user to be supported by the system. Define a different integrity category for each such role.

Examples: shipping clerk, cash teller, inventory manager.

P2. Identify the different types of "Controlled Data Items" (in the Clark & Wilson scenario) and define a different integrity category for each type; none of these categories can be the same as any of those defined in P1. Note that one or more integrity categories should also be defined for system and administrative software.

Examples: accounts receivable, cash account, inventory, general ledger, audit trail, production software, TCB, authorization tables.

P3. Identify each type of administrative user who is expected to exercise some kind of authority. Define a different integrity category for each type; none of these categories can be the same as any of those defined in P1 or P2.

Examples: security officer, system administrator, system operator, DP manager, programmer, tester.

P4. Identify which roles are conflicting. When a user is registered with the system, or given a new job assignment, he would be authorized for ("cleared for") only a set of non-conflicting roles, usually only one. The ability to modify any "Controlled Data", especially authorization tables or system software, would in general inherently be a conflict of roles: no

user will ever be given (direct) authority to modify any "Controlled Data" since rules P1 - P3 insist on three disjoint sets of integrity categories.

P5. After a transaction, or collection of related transactions, has been programmed and tested (on non-production data) the operational copy of the transaction's software and any internal data that the transaction relies upon is given the "production software" integrity category marking to prevent unauthorized changes to it. (No person can ever log on with the "production software" category).

P6. After the transaction has been certified to be functionally correct (well-formed, generates the required audit trails, performs double entry if appropriate, etc.) the transaction is established as a partially trusted subject whose v-min integrity label is the role a user must be in to exercise the transaction and whose a-max label is the set of Controlled Data Item types it is allowed to create or modify.

P7. It is assumed there are one or more "security authorizing" transactions (trusted subjects) that must be exercised by, and only by, appropriate security officers, system administrators, or DP managers to register users and to perform steps P5 and P6: P4 prohibits them from directly performing those actions (i.e., through untrusted system utility software.) This is a version of the "who watches the watcher" problem and is discussed in more detail below.

P8. The responsibility for creating, installing, running, and acting on the results of IVP (consistency checking) software cannot be assigned to the system itself in any general way -- it must be done by management. The system can, however, ensure that results that purport to come from an IVP (such as a reconciliation of accounts) cannot be generated by masquerading software if each IVP (or collections of them), and only an IVP, is installed (by procedures like P5 and P6) as a trusted subject whose a-max label is something like "account reconciliation."

5. Comparison with Clark & Wilson Model

Following, very briefly, is an analysis of how each of the 9 requirements of the Clark & Wilson model is met by the above configuration rules or by the inherent properties of a B1 or higher class trusted system.

C1: IVP Certification -- P8

C2: TP Certification -- P6

E1: TP to CDI enforcement -- P2, P4, P6, and TCSEC security (integrity) policy enforcement

E2: User to TP enforcement -- P1, P4, P6, and TCSEC security (integrity) policy enforcement

C3: Separation of duties -- primarily P4 and P1

E3: User identity authentication -- TCSEC identification and authentication

C4: Audit trail -- P6 and TCSEC audit requirements (for security actions)

C5: Valid data -- P5, P6

E4: Can't change own authorization or affect own transaction -- P4, P7, and TCSEC security (integrity) policy enforcement

6. Who Watches the Watchers?

One topic that was somewhat glossed over above is that of what kind of controls should be exercised over those who are entrusted with authority to change the system itself, including authorization information such as the integrity categories given to users or trusted subjects. The following rules, all derived from the separation of duties principle, seem to make sense:

R1. No user (including, for instance, a security officer) may change his own security authorization (e.g., set of roles).

R2. No user may authorize the installation of or modification of a transaction he can exercise nor can he modify its security authorization (a-min and v-max).

R3. No user may authorize the installation of any transaction he is not competent to understand or does not have jurisdiction over (in some sense), nor may he set its security authorization.

R4. No one may alter or destroy any audit records.

Rules R1, R2, and R4 could be easily built into the security authorization transactions of P7, mostly to prevent a security officer from giving himself inappropriate integrity categories.

R3 is both more subjective and dependent on an organization's management style. In some ways it is related to the question of "who is authorized to certify that a given trusted system is allowed to operate over such and such a range?" One possibility would be to define a "transaction certification table" that defines for each expected combination of (user-role, set of CDI-types) what kind of authority (e.g., administrative role, which indicates degree or nature of responsibility) a

person must have to set up labels on a transaction (trusted subject) to that set of values. The details probably don't matter greatly provided some form of rules R1-R4 is built into P7 and certified to be there by some kind of two-person rule built into the initialization portions of the system.

7. Relevance of TCSEC Classes and Features

Apart from supplying many of the features needed to support the policy discussed here, a system built to and evaluated against the TCSEC has the following additional virtues:

B1: (minimum class required in order to support labels) -- the basic enforcement mechanisms of the system have been tested by an independent organization and the documentation meets a certain level of quality

B2: The integrity policy is enforced over all information and devices in the system. This means that the chances of error or deliberate subversion permitting a user to perform an unauthorized action are extremely low; it also means that critical data (e.g., an order to buy a million shares of IBM) delivered to a user comes with great assurance only from a properly authorized source; this would be especially so if the principles of the B2 or higher sections of the TNI were used to evaluate the network portions of a system.

B3:

A1: Still greater confidence in the ability of the system to enforce all the rules. Whether a system of these high evaluation classes is required in any given installation would be a judgement of the system's owner. However, the principles of the Yellow Books (environment guide) [DoD85b] might be adaptable if some means could be found to associate different integrity categories of data with different sensitivity levels and a notion of "risk range" developed.

8. Miscellaneous Topics

Here are some afterthoughts that should be explored further.

a. It should be made clear that a subject can be trusted (even partially) only if it is started in a known state, e.g., at a specified entry point, with specified initial values for some data. Except that a trusted subject is itself very analogous to an entire trusted system (or vice versa) and thus might be usefully modelled in terms of secure initial states and security-preserving state transformations this principle doesn't seem to fit too well into, say, the original Bell & La Padula model [Bell75]; it may call for some additional rules in the model in a B2 or higher class system. In any case, one might

have to explicitly define a trusted subject as an entity that has more internal structure (e.g, the code it is to execute) than the uninterpreted subjects of the original Bell & La Padula model.

b. A system that enforces the Biba model has what on first glance appears to be an inherent complication: any data, including the executing code that a subject reads must have an integrity label that is a superset (dominates) either of the subject's labels. Thus, for instance, any code or data (other than any in the output CDI's) that a transaction to be run in, say, "cashier role" needs to access must contain "cashier role" in its label. On second glance this actually makes sense: such a transaction is certified to operate properly only if it takes data from a cashier and no-one else; any data that does not contain the "cashier role" marking could have been modified (supplied by) some other kind of person. (Note that "operates properly" here includes the notion that it is inherently improper for a transaction to accept data from anyone other than one of its authorized users.) In any system there are large portions of software and read-only data that a transaction program needs to use and can in general rely on since they are supplied by the manufacturer and "certified" through use.

Rather than having the security officer mark such data so that it can be read (contents trusted by) all transactions it would be useful to have a limited notion of hierarchical integrity categories: TCB dominates vendor library dominates production software dominates {any user role}. (It is the "dominates {any user role}" that is not strictly hierarchical: the roles themselves have no dominance relation with each other.) Note that the appendix in [Ware] suggests such a concept for security categories so the idea is not contrary to the principles of the TCSEC, although it is not clear that anyone has implemented it.

c. The (partially) trusted subject as formulated in the model above cannot do one thing that might make sense: it cannot, say, be allowed to read data of type cashier or type teller or type funds transfer but be constrained from (not allowed to be executed by) reading any other kind; it must be one or all (which however could not have been written by any person limited to only one of the three roles.) One logically consistent extension to allow this would be to redefine what the v-min label on a subject is: it would be a set of sets of categories; any data which dominates (whose label is a superset of) any of the sets in v-min could be read. The informal meaning of this would be that no matter which of a limited set of sources the subject got its information from it could be relied upon to properly create or modify the data it is constrained to operate

upon. But that begins to be complicated and would certainly increase overhead[4].

Another alternative is relatively straightforward: if a trusted subject needs to deal with several different types of data it must do so through separate intermediary "agent" trusted subjects for each type. For example, if subject X, which updates data of type C, needs access to data typed A (i.e., whose integrity label is A) and type B, but no others, a new "disjoint" integrity class, say, "A or B" would be defined and two subjects S and T created -- S would access A on X's behalf, T would access B. The labels on the subjects would be

a-max(S) = a-max(T) = v-min(X) = A or B;
v-min(S) = A; v-min(T) = B; a-max(X) = C

Both alternatives have their strengths and weaknesses and need more thought.

d. As noted in [Bell86] the concepts of v-min and a-max replace the [Bell75] concept of "current security level." (In our case, integrity.) The example discussed in this paper has no need for a subject to change its current level, so that aspect of a possibly desirable system behavior is not modelled. To do so would probably entail adding two more labels -- a "true" alter-maximum and view-minimum -- renaming what we have here as current-alter-maximum (c-a-max) and current-view-minimum (c-v-min), and adding appropriate rules constraining how a subject may change c-a-max and c-v-min given the values of a-max, v-min, and what the levels of the currently accessible objects are.

e. It is recognized that the model presented here does not address the issue of subject-to-subject interactions other than through shared objects. Without further refinement to permit controlled domain crossings (e.g., whereby a process changes its subjecthood by moving one domain to another) the only way one subject (such as a user) could cause another (such as a transaction) to be invoked would be through the change in the value of some data object that could be written by the first subject and read by the second[5]. Note that for a transaction to synchronize with an invoking user there would have to be a

---

[4]It would however allow the integrity *-property to be simplified by deleting the explicit ability to read anything a-max dominates: if the subject is be able to view objects of type a-max as well as modify them, that category set would have to be expressly included as one of the sets in its v-min.

[5]Such an object could readily be constructed since the purpose of a transaction is to read information supplied by a user, i.e., a-max(user) = v-min(transaction).

corresponding reverse flow of information; as the model is given here there is, quite properly, no means for that flow since user-subjects are not supposed to be able to read any data written by transaction-subjects[6]. The ideas in paragraph b or c above could be used to define an integrity category for, say, semaphores that would be accessible to all subjects and thereby permit synchronization. Note in any case that the problem of synchronization between mutually distrustful subjects (which is in effect what a separation of duties principle mandates) is made quite explicit by this model and whatever solution is adopted will have to be thought through carefully: although there seems to be no intuitive integrity dual to the covert timing or storage channels addressed in enforcing a mandatory non-disclosure policy, that seems to be what we are dealing with here.

f. There is a possibly superficial difference between the commercial world and the NSE world that could be fundamental: the latter assumes, more or less, that its users are trustworthy to the extent of their clearances ("need-to-know" and compartmentation being the primary vehicles for dealing with human imperfections). The commercial world, however, in insisting on the separation of duties principle inherently assumes it users are untrustworthy. The implications of this difference, if indeed there really is one, need to be explored.

g. Again, it is assumed that the reader is generally familiar with all the topics discussed. This is not intended as a tutorial on mandatory integrity policies, of any variation, or of the TCSEC as it relates to them, nor is it intended to recapitulate [Clark].

h. [Boebert] claims that a Biba policy cannot support an integrity-enforcement example ("assured pipeline") that seems very similar to the Clark & Wilson requirements. This is true if all one has is integrity levels. But if one uses integrity categories and (partially) trusted subjects it is possible to establish a protection regime that is very similar to that of the SAT type and domain mechanisms. His secure labeller example could be set up in the following way. There would be three subjects: User, Labeller, and Output; and three integrity categories: U for Unlabelled, L for Labelled, and P for Printer (that which Output, and only Output, is to write to; Boebert's example doesn't include the printer as an object.) The categories would be assigned to the subjects as follows:

v-min(User) = a-max(User) = U
v-min(Labeller) = U

---

[6]Because the data they deal with are of disjoint integrity categories.

a-max(Labeller) = v-min(Output) = L
a-max(Output) = P

Note that all three subjects would be untrusted with respect to the disclosure policy, i.e., all would run at the security level of the data being output; the User is also untrusted with respect to integrity. It is an exercise for the reader to demonstrate that the above assignment of integrity categories gives precisely the same access matrix as the DDT (Domain Definition Table) on the lower right quarter of p. 25 of [Boebert][7].

9. Acknowledgements

Credit must be given to my colleagues David Bell, Marv Schaefer and Steve Crocker for some intense, but totally fortuitous (insofar as the topic of this paper is concerned), conversations that occurred late in October about the notion of partially trusted subjects in the context of mandatory security policies. Until they read the first drafts of this they may not have realized how germane the concept is to the "commercial security policy" problem or what they had unintentionally inspired. The first draft of this paper was written as an unsolicited position paper for the "Workshop in Integrity Policy in Computer Information Systems" held Oct. 27-29 at Bentley College in Waltham, Mass., to "respond to the challenge presented" by [Clark], so the organizers of that workshop must also be given credit for having provided the impetus.

Credit must also be given to Steve Lipner's seminal paper [Lipner] on this topic: however I must confess that although I did attend its presentation in 1982 it appears that I never actually studied the paper until after writing my first draft last summer and in fact I very erroneously and unfairly remembered his formulation as not using integrity categories, which it most definitely does.

10. References

[Bell75] D.E. Bell and L.J. La Padula, Secure Computer Systems: Unified Exposition and Multics Interpretation, MTR-2997, The Mitre Corporation, Bedford, Mass., July 1975. (ESD-TR-75-306)

---

[7]At this point Boebert would be justified in raising the objection that his DTT (Domain Transition Table) is not reflected in any aspects of the model here. That would come out of resolving the subject-to-subject transition issues raised above in paragraph e.

[Bell86] D.E. Bell, "Secure Computer Systems: A Network Interpretation," Proceedings of the Second Aerospace Computer Security Conference: Protecting Intellectual Property, Dec. 2-4, 1986, McLean, VA, pp. 32-39.

[Biba] K.J. Biba, Integrity Considerations for Secure Computer Systems, USAF Electronic Systems Division, Bedford, MA, ESD-TR-76-372, April 1977.

[Boebert] W.E. Boebert and R.Y. Kain, "A Practical Alternative to Hierarchical Integrity Policies," Proceedings of the 8th National Computer Security Conference, Sept. 30 - Oct. 3, 1985, Gaithersburg, MD, pp. 18-27.

[Clark] 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 27-29, 1987, Oakland, CA, pp. 184-194.

[Courtney] R.H. Courtney, Jr., "An Industry View of the DoD Computer Security Center Program," Proceedings of the Sixth Seminar on the DoD Computer Security Initiative, Nov. 15-17, 1983, Gaithersburg, MD, pp. 11-13.

[DoD85a] Department of Defense National Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, Dec.1985, DoD 5200.28-STD.

[DoD85b] Department of Defense National Computer Security Center, Security Requirements: Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, CSC-STD-003-85, June 25, 1985.

[DoD87] Department of Defense National Computer Security Center, Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-005, July 31, 1987.

[Lipner] S.B. Lipner, "Non-Discretionary Controls for Commercial Applications," Proceedings 1982 IEEE Symposium on Security and Privacy, April 26-28, 1982, Oakland, CA, pp. 2-10.

[Murray] W.H. Murray, in "Panel Session -- Base Spectrum of Computer Security Requirements," Proceedings of the Sixth Seminar on Computer Security Initiative, Nov. 15-17, 1983, Gaithersburg, MD, pp.14-16.

[Ware] W.H. Ware, ed., Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security, Rand Corporation, Santa Monica, CA, R-609-1, October 1979 Reissue.

A8

THOUGHTS ON THE CONCEPT OF INTEGRITY AND INTEGRITY POLICIES

by

ROBERT H. COURTNEY, JR.

The following ideas were conveyed to me (Stuart W. Katzke) by Bob Courtney through private communications. Since I felt strongly that Bob's thoughts should be shared with others, I requested (and Bob granted) permission to assemble and (slightly) edit his comments for inclusion in this Workshop Report. Bob's comments follow.

With regard to the concept of an integrity policy, I still have a horse/cart relationship problem. Let me suggest several things the resolution of which appear to me essential to real progress in this area. They are these:

1. What do we mean by "integrity" in the context in which it is used? How can you decide what to do about it until it is satisfactorily defined?

2. What is a general integrity policy? You need to be able to state what it is and why we need it before we pursue it. What is done about integrity varies dramatically from application to application within a single organization. Why is the pursuit of policy the correct approach? Why isn't the whole business simplifiable to identification of the requisite means for the attainment of the desired degree of integrity in each specific application, assuming we have managed to define it and can determine how much and what kind we need? I am very much afraid that pursuit of policy will ultimately derail the whole potentially valuable activity now underway. What is magic about the word "policy" in this context? I am very much afraid that policy at this level implies intellectual rigidity and this is no time for that.

3. I caution against the notion that an integrity policy should be implemented in vendor products. I cannot imagine that you will ever be able to identify and implement an integrity policy which is sufficiently constrained to permit its implementation in any reasonable and economically feasible manner and yet so broad as to make it useful in the vast diversity of applications made of general-purpose data processing products. Let me offer some further thoughts on this integrity issue. I don't think that real progress will be made until most of the people considering the matter have a common understanding of what it is we are talking about. The following comments reflect what I think we mean, or should mean, when we talk about data integrity.

Integrity is a fundamental virtue and, like other fundamental virtues such as honesty, cleanliness, and charity, its definition is very context-sensitive. Just as we cannot effectively pursue those other virtues without defining them within a particular context, neither can we simply pursue integrity except as we adequately define that context.

When we try to write policy about personal integrity, we are forced to say what we mean by it within the context covered by the policy. We must specify the specific ramifications of it which we want to address and then state what must be achieved in each of these. I believe that you can no more have a data integrity policy than you can have a personal integrity policy except as that policy addresses the particular components of integrity about which we are concerned in a particular environment.

The specific requirement for data integrity will often, even generally, vary with the use to be made of those data, that is, from application to application. If I am using a local telephone number to tell me in what general area a person lives, then I need accuracy in only the first three digits. Errors in the remaining 4/7 of the number will not bother me. On the other hand, if I am trying to call, errors anywhere in the number render the whole number useless. The accuracy requirements for these two applications is quite different. The best you will be able to do with an integrity policy, then, is to say that the data should be as good as they need to be for the purpose at hand. That offers little guidance in what to do about it.

Let's take the problem apart and look at it. (Data) integrity has two equally important components. They are (1) data quality and (2) assurance of that quality. Data quality cannot be relied upon in the absence of means to verify it.

Data quality consists of five elements. They are:

1. Accuracy,
2. Completeness,
3. Precision,
4. Timeliness, and
5. Confidentiality.

If there are deficiencies in any of these five elements, the resulting integrity of those data may not be as great as might be desired in a perfect world, but those deficiencies may not, and usually do not, constitute fatal flaws unless we are unaware of them. The sine qua non of integrity is freedom from unpleasant surprise.

It is quite apparent that all five elements will not be of equal importance in any specific application of the data. In

some cases, one or more of them may be of relatively little or no importance at all. In others, they may be essential to the integrity and therefore the usability of those data.

We are quite accustomed to seeking integrity in our systems design. We virtually never call it that because the term used there, as with data integrity, lacks specificity. But we do indeed pursue freedom from key deficiencies and we also seek the other component - quality assurance, in this case, awareness of system deficiencies which have significant bearing on our ability to use the system for a specific purpose.

Summarizing the above: when we speak of data integrity we should be addressing both data quality and assurance of that quality. The former is seriously impaired without the latter. There are five elements which describe quality, as listed above, and each can be addressed in at least a semiquantitative way. But, again, the other component of integrity, quality assurance - an understanding of the degree to which the data may be deficient in any of the five quality elements - is no less important to integrity than is quality.

I strongly suggest that folks not waste time pursuing the notion of integrity policy but instead concentrate on (1) assessing the measures which should be applied to the identification of the need for data quality, (2) recommendations of a systematic means for the application of measures necessary to the attainment of that quality, and, (3) to the identification and recommendation of practicable means for assessing data quality.

Finally, there is one thing which should be realized by all participants. No new approach to integrity will be acceptable if it forces obsolescence of existing application code. This does not mean that a new higher plateau of integrity must be achievable with old code; it must be able to cohabit with old code. Then new levels of integrity can be achieved in each application for which new code is written or, and far better, for which old code is upgraded.

Any approach which forces obsolescence of existing application code would create more integrity problems than could ever be solved with that new approach.

No major vendor can be really supportive of an approach which obsoletes his customers' application code because the customers will think that he is out of his mind. Those customers have multibillion dollar investments in that code.

There is always danger, as you know as well as I do, that companies may be represented on any committee by starry-eyed techies who have never been in touch with the realities of the

business and who can enthusiastically endorse anything which is intellectually titillating and leave the impression that the whole enterprise from which he comes is behind him. Which is where they often are - so far behind him that he is not even in the same parade.

A9 IMPLEMENTING THE CLARK/WILSON INTEGRITY POLICY USING CURRENT TECHNOLOGY

William R. Shockley

Gemini Computers, Incorporated
P. O. Box 222417
Carmel, California 93922
(408)373-8500

Abstract

The security policy presented in [1] is viewed as expressing a set of valid requirements that must be enforced with a high (Class A1) level of assurance. A policy expressed in terms of a lattice of access classes is formulated that (together with discretionary and supporting controls) implements the requirements, leading to the conclusion that current Trusted Computing Systems based on the Bell and LaPadula model [2], in combination with well-understood extensions of this model, such as the Biba strict integrity policy [3] and Schell and Shirley's program integrity policy [4], are capable of supporting the Clark/Wilson policy.

Introduction

The material presented in [1] by D. D. Clark and D. R. Wilson has received a relatively high degree of attention as an accurate representation of what the business community means by the term integrity, when used in the context of computer security. It is proposed by Clark and Wilson, that this policy should be enforced with the level of assurance normally associated with the enforcement of the Mandatory Access Control Policy in the Trusted Computer System Evaluation Criteria [5] for trusted systems useful to the general data processing community. Clark and Wilson also state, as their two primary conclusions, that "a lattice model is not sufficient to characterize integrity policies", and that "distinct mechanisms are needed to control disclosure and to provide integrity".

In this paper, issue is taken with both of these conclusions: at least, with the implication that might be drawn from them that "therefore, lattice policies are not useful in this context". The premise upon which this paper is based is that an appropriate combination of lattice policy based controls and supporting policies tailored to fit are quite capable of enforcing the policy implicit in the Clark/Wilson requirements and that, indeed, TCB products available now, or in the near future, are capable of meeting these requirements fully.

What will be presented in this paper is an abstract description of a system compliant with the requirements, using a lattice of access classes expressed in terms of integrity and disclosure categories, together with various constraints upon user clearances and the subjects which can be activated upon the user's behalf. The system of access classes represents the non-discretionary component of the requirements, while the enforced constraints upon the clearances a security officer may enter, together with constraints upon what subjects may be activated, represent supporting and discretionary aspects of the requirements. By "non-discretionary" we mean a policy concerned with global and persistant constraints upon information flow: as Denning has shown [7] that every such consistent policy can be represented in terms of a lattice of access classes, our insistance on sharply separating "non-discretionary" policies from everything else is well-grounded in theory.

The analysis presented below was pursued in conjunction with the SeaView project to design a Class A1 DBMS for implementation in the near term. The project is sponsored by Rome Air Development Center (RADC), and is being pursued by a team consisting of scientists from SRI International and Gemini Computers, Inc. The relevancy of the Clark/Wilson requirements to this project has been recognized since the paper was presented, although we take issue with some of the conclusions presented in [1]: military, as well as commercial data processing centers, are interested in controls that allow enforcement of separation of duty and well-defined transactions with high levels of assurance. It is our current intent to implement a working demonstration of concept on a Class A1 certifiable security kernel (GEMSOS) within the next six months. The demonstrated system will also be compliant with the most important of the Clark/Wilson requirements, using techniques similar to those discussed below.

The abstract design discussed below makes use of several refinements to the basic Bell and LaPadula model, namely, the strict integrity policy formulated by Biba [3] and its extension to trusted subjects by Schell and Shirley [4], called the program integrity policy. As these refinements may not be familiar to some readers, a brief description of them is provided in the next section. The abstract design presented in this paper may be regarded, on the one hand, as a re-interpretation of the Clark/Wilson requirements in more familiar terms, and on the other, as providing a worked example shedding light on the usefulness of both strict and program integrity, in conjunction with the more familiar disclosure policy specified by the TCSEC [3].

Because the association of the term mandatory with that access control policy using a lattice of partially-ordered sensitivity labels to represent both user clearances and

information sensitivity has been criticized in [1] (although this usage is taken as an axiom in [5]), the original term used by Saltzer and Schroeder in [6], non-discretionary will be used throughout this paper to signify access control policies based upon labels partially-ordered by a dominance relation.

Technical Background

The Clark/Wilson Requirements

For completeness, the rules given by Clark and Wilson for a commercial integrity policy are restated below. Detailed arguments for their content are provided in [1].

Definitions

Constrained Data Item (CDI) -- those data items within the system to which the integrity policy must be applied.

Integrity Verification Procedure (IVP) -- a procedure, the purpose of which is to confirm that all of the CDI's in the system conform to the integrity specification at the time the IVP is executed.

Transformation Procedure (TP) -- a well-formed transaction that changes the set of CDIs from one valid state to another.

Unconstrained Data Item (UDI) -- a data item not covered by the integrity policy, (in the sense that it represents new non-validated information that has just been input into the system).

Enforcement rules.

Those rules that represent requirements imposed upon human certifiers are labeled C1 through C5, while those to be enforced primarily by the automated system are labeled E1 through E4. Comments have been added to some of the rules indicating those for which a design will be only sketched in this paper.

C1: All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run.

Comment: For simplicity, in this paper IVPs will be treated as ordinary TPs that read, but do not modify, a set of CDIs. The treatment of IVPs as distinct from TPs would add unecessary complexity to the discussion without changing the abstract design substantially.

C2: All TPs must be certified to be valid, that is, they must take a CDI to a valid state, given a valid initial state. For each TP the certifier must specify a list of

CDIs (called a relation) which the TP has been certified to manipulate correctly.

Comment: For simplicity, it will be assumed that a TP both reads and writes every CDI it "manipulates". The addition of controls to enforce a distinction between those CDIs that are "read-only", those that must be both read and written, and those that are "write-only" (append), is interesting and possible, but leads to substantial additional complexity.

E1: The system must maintain the relation referred to in rule C2, and must ensure that the only manipulation of any CDI is by a TP, for which the CDI occurs in the relation for the TP.

Comment: The relation will be encoded within the access class of the object containing the TP.

E2: The system must maintain a list of relations associating triples of the form <UserID, TP,CDI> that identifies which users may cause which TPs to be executed to manipulate which CDIs. (Note that each CDI must be in the relation for the given TP, but not all such CDIs may occur for a given UserID and TP).

Comment: This requirement will be supported by maintaining a database within the TCB that is consulted whenever a subject executing a TP is activated on behalf of an authenticated user. Limitation of the CDIs available to that subject will be enforced by encoding a list of CDIs in the access class of the subject, so that it cannot be subverted by malicious code (even in the TPs), once the subject is activated.

C3: The list of relations in E2 must be certified to meet the separation of duty requirement.

Comment: The abstract design will provide the potential for automated support for this requirement, although it is listed by Clark and Wilson as a requirement to be enforced by the human certifiers alone.

E3: The system must authenticate the identity of each user attempting to execute a TP.

Comment: Implementation of a system upon an evaluated TCB meets the intent of this requirement, as a Class C2 or above TCB includes an evaluated identification and authentication capability that cannot be bypassed.

The basic functionality assumed for the abstract design is that the authenticated user may select either to execute an untrusted subject (for creating programs, entering UDIs, and the like) or a subject that executes a single TP before returning to the trusted logon menu. The controls implied by Rule E2 are enforced by the trusted logon routine, which is part of the TCB and therefore evaluated for correctness. This routine consults the triple database to evaluate the TP and CDIs selected by the user for compliance with rule E2 before activating a subject to execute the TP on the user's behalf.

C4: All TPs must be certified to write to an append-only CDI (the log) all information necessary to permit the nature of the operation to be reconstructed.

Comment: A conventional TCB provides substantial built-in and unbypassable support for this audit function, relieving the certifier from having to check that a TP is self-policing with regard to security-relevant actions. This rule, for the sake of simplicity, will not be discussed further.

C5: Any TP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI.

Comment: Although primarily a rule for certifiers, the discussion below will show how the execution of such TPs are supported.

E4: Only the agent permitted to certify entities may change the list of such entities associated with other entities: specifically, those associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

Comment: This rule will be enforced primarily by the module of the TCB that allows the security officer to enter clearances -- if a given user (represented by a unique UserID) has one type of clearance (as a "certifier") that user will not be allowed to have a "TP user" clearance, and vice versa.

Non-Discretionary Policies

The following information characterizes those elelments of non-discretionary policies that will be used in the remainder of the paper. Because these mechanisms are well-known, extensive commentary is not provided: this section is intended only to precisely identify the access control rules enforced by the underlying security kernel. At least one such system,

certifiable at Class A1 (GEMSOS), that enforces rules exactly as stated below, is commercially available.

Disclosure Classes

There is a set D of disclosure classes, partially ordered by a dominance relation <= , such that if d1 <= d2 in D, d2 is of equal or greater sensitivity (relative to disclosure) than d1.

Strict Integrity Classes

There is a set I of integrity classes, partially ordered by a dominance relation <= , such that if i1 <= i2 in I, i2 is of equal or greater sensitivity (relative to modification) than i1.

Access Classes

Consider the Cartesian product A = D X I of access classes. Dominance is defined for access classes such that for any a1 = <d1, i1> and a2 = <d2, i2>, a1 <= a2 if, and only if, d1 <= d2 and i2 <= i1.

It is noteworthy that dominance for access classes is based upon dominance for the integrity class components in the inverted sense: that is, given identical disclosure classes, two access classes of differing integrity components are ordered from more to less sensitivity relative to modification. The utility of this convention is that it simplifies the expression of the non-discretionary policy rules themselves. (Fundamentally, a1 <= a2 means that information flow from class a1 to class a2 is permitted by both the disclosure and strict integrity policies simultaneously.)

Basic Combined Policy

The basic combined policy is stated in terms of active subjects, entities that cause information to flow between objects, objects, passive data repositories that are manipulated by subjects, and two access modes, read and write. It is assumed that a subject that has read access to one object and write access to another, can transfer information derived from the first to the second.

Each subject is labeled with two access classes, a readclass and a writeclass, such that writeclass(s) <= readclass(s) for all subjects. If writeclass(s) = readclass(s), the subject is said to be untrusted, if writeclass(s) < readclass(s) the subject is said to be trusted.

Each object is labeled with a single access class, taken to represent the combined disclosure and modification sensitivities

of data that may be written into that object. The basic policy regarding those objects a subject can access for read and write is:

A subject s may obtain read access to an object o only if class(o) <= readclass(s).

A subject s may obtain write access to an object o only if class(o) >= writeclass(s).

Note that these rules are identical, for untrusted subjects, to those in the Bell and LaPadula (because writeclass(s) = readclass(s), so that the subject has, in effect, a single label). The rules represent a convenient refinement of the rules for Bell and LaPadula trusted subjects -- in the Bell and LaPadula rules, a trusted subject may "write down" to any object, whereas for the rules given above, the privilege to "write down" is here limited by the subject's writeclass.

The stated rules also merge traditional non-discretionary disclosure policies with the Biba strict integrity policy, by combining the two arbitrary partially ordered sets of labels into one and redefining the dominance relation appropriately.

Program Integrity

Because we are providing for trusted subjects as an explicitly recognized model component, it is convenient to recognize execute access explicity, in order to preserve the desired semantics for integrity labels. (This will be explained more fully below.) The policy for achieving this is called by Shirley and Schell program integrity. Its purpose is to ensure that a trusted subject may only execute objects that are of the same integrity as the maximum integrity the subject can modify. The rule given here extends this idea by including a constraint that the subject may only execute objects that are of the same secrecy as the minimum secrecy the subject can modify. A brief rationale for the rule is given in the next subsection.

A subject s may execute an object o only if
integrity(class(o)) >=
integrity(writeclass(s))

(The dual is also enforced but will not be discussed further).

A subject s may execute an object o only if
disclosure(class(o)) <=
disclosure(writeclass(s))

When the subject is untrusted, (i.e., readclass(s) = writeclass(s)) program integrity degenerates to strict integrity,

where execute access is considered a variant of read access. The program integrity policy is a genuine refinement to Biba strict integrity for subjects trusted with regard to integrity, because in this case there might exist objects that the subject can read, but not execute.

Semantics of Strict and Program Integrity

The rules given above treat "integrity" as a formal abstraction that is the mathematical dual of the more familiar non-discretionary policy for disclosure. In order to make use of the policy in a meaningful way, one must give an intended semantics for the integrity dominance relation: under what circumstances does it make sense to say that some information has "more" integrity than some other information? What does it mean to give a program (which is data of a special sort) an integrity label? The semantics defined here is an interpretation tailored for use in a Trusted Computing System, in which information is created or modified by a subject (trusted or untrusted) executing a program.

The fundamental notion is that for a program to be given an integrity class (other than, perhaps, the lowest) it must be correct: that is, it is believed, for some reason, to correctly transform its input to its output. Typically, its correctness is based upon some software validation technique, or combination of techniques. Moreover, for a given application, the data managed by the system may be more or less critical: for instance, some data, if corrupted, might result in the loss of life while others might be administrative in nature. The sponsor of a system, we can imagine, will group the data into collections of data that are partially ordered by criticality, and for each group, establish those software validation techniques that must be applied in order to trust a particular program to minipulate data of a given criticality. The resulting system defines the particular integrity policy to be enforced.

The rules enforced for the basic security policy (as applied to untrusted subjects) work in a way that is intuitively correct, for the semantics of strict integrity given above. A subject with integrity class I may modify only objects of integrity I or below, and may read (and thus, execute) only objects (programs) of integrity I or above. Thus, every program used to modify an object of integrity I, is correct, in the sense that it has met at least the software validation requirements for integrity I. Moreover, the input data upon which the modified data I depends is also of integrity I or above: thus, the desired constraint (that data not be modified by programs that have not been subjected to the required validation) is met both directly and indirectly.

We may now discuss what it means for a subject to be "trusted" with regard to integrity. The basic non-discretionary policy allows a subject trusted with regard to integrity to modify data of high integrity, while reading data of lower integrity. The program executed by such a subject must be more than just "correct" (in the sense that it transforms input to output correctly) -- it must be able to distinguish low from high integrity input and make its modifications to high-integrity data accordingly. What this means varies from circumstance to circumstance. In the simplest case, the program might be required to be "flow-free" -- that is, it recognizes the integrity class of the input data and modifies only data of the same integrity. A more complex case involves an "integrity upgrade", in which ostensibly low-integrity data is validated and copied to a high-integrity repository.

The need for the program integrity policy should now be apparent: because a subject trusted with regard to integrity may, from time to time, modify data at the (high) integrity of its writeclass, it must only use programs of appropriately high integrity so that it never produces high integrity data using programs that have not been appropriately validated.

Integrity/Disclosure Policy for Clark/Wilson Requirements

We turn now to a non-discretionary policy for the Clark/Wilson requirements expressed in terms of a combined, partially-ordered set of access classes of the form <integrity class, disclosure class>. For the sake of simplicity, we will use only integrity and disclosure categories. (No claim is made that the solution below is unique, or even that it is the best one: it has been selected for ease of exposition). Recall that we are treating IVPs as special kinds of TPs, and that we are assuming that all CDIs referenced by a TP must be accessible for both read and write access. However, we will label data to the granularity of individual CDIs and TPs in order not to unduly trivialize the problem.

An integrity class consists of an arbitrary subset of the set of all integrity categories, and will be denoted [a, b, c, . . .] using square brackets, where a, b, c and so on are integrity categories. Similarly, a disclosure class consists of an arbitrary subset of the set of all disclosure categories, and will be denoted {x, y, z, . . .} using curly brackets. An arbitrary access class will typically be denoted:

[a, b, c, . . .]{x, y, z, . . . }.

The dominance relation for access classes is defined in the usual way: an access class A1 = [a, . . .]{x, . . .} is dominated by an access class A2 = [b, . . .]{y, . . .} if, and only if, [a, . . .] is a superset of [b, . . .] and {x, . .

.} is a subset of {y, . . .}. It is easy to show that the integrity classes form a lattice, that the disclosure classes form a lattice, and that the combined acccess classes form a lattice composed by taking the Cartesian product of the two and inverting the integrity dominance relation as previously suggested. Thus, for A1 <= A2, A1 carries at least all of the integrity categories of A2 and at most all of the disclosure categories of A2.

Assignment of Access Classes to Clark/Wilson data objects

Each CDI will be assigned a "generic" integrity category, [c], as well as an individual integrity category of its own: thus, the access class for (say) CDIi will include integrity category [c, CDIi]. This assignment reflects the idea that the CDI can only be manipulated by TPs (program objects) that have been certified to be "correct", but allows what "certification" means (relative to correctness) to vary from CDI to CDI. The integrity category [c] may be interpreted as meaning that every CDI has some common level of "correctness", over and above none at all.

Each CDI will also be assigned its own disclosure category as well as common disclosure category {c}. (We might note at the outset that the Clark/Wilson requirements imply a minimal policy for disclosure as well as integrity as Rule E1 implies that a TP must not be allowed to manipulate a CDI it has not been certified for -- which means that it must be allowed to read only the CDIs assumed readable during the certification.)

Thus, the combined access class for (say) CDIi will be:

CDI object: CDIi
access class [c][CDIi]{CDIi}.

Each TP is itself a data object, and has been "certified" to correctly manipulate some list of CDIs. The TP will thus be given an integrity class that includes the integrity category for each of those CDIs. Its certification must include all of the validation tasks required for any of the CDIs it can manipulate. In addition, we suppose there is a common set of validation requirements to be met, that are represented by the common CDI category [c]. Finally, the certified TP is correct in a sense that an ordinary CDI is not: it is executable. This attribute will be represented as a common TP integrity category [t]. As for CDIs, we also give each TP an individual disclosure category as well as disclosure categories {t,c}. These will be used to prevent TPs from being read (and therefore executed) by users not authorized to execute TPs at all.

TP object: TPj
manipulates: CDIa, CDIb, . . .

access class: [t, c, CDIa, CDIb, . . .][TPj]
{t, c, TPj}

A UDI is a data item that is assumed to have no integrity at all. Obviously, it can then be neither a TP nor a CDI. Note that programs under construction (not yet certified) fall in this category. We wish to prevent most subjects and some users (TP users) from executing or examining programs under construction, and we wish to prevent some users (certifiers) from creating or modifying programs under construction. These desires suggest giving UDI objects a generic integrity and a generic disclosure category of their own:

UDI object: UDIk
access class: [u]{u}

The meaning of integrity category [u] is "not created by non-programers".

An observation that may not be apparant on first reading is the relation between the combined access class of a TP and the CDIs it manipulates: these access classes are always non-comparable, because the TP has a disclosure category the CDI does not (its unique disclosure category) and vice versa. However, the TP has strictly greater integrity than any of the CDIs it manipulates, because it carries all of their unique integrity categories as well as the integrity category [t]. Similarly, any CDI has greater integrity than any UDI: however, no CDI or TP has a combined access class that is comparable to any UDI when both components are considered. What has been described so far is a system in which the individual objects of interest (CDIs and TPs) are in isolated domains, but are "tagged" with access classes that encode their attributes relative to the Clark/Wilson rules. In particular, the integrity component of the access class of a TP serves as the "relation" required by Rule E1.

Non-Discretionary Policy

The non-discretionary component of the policy implicit in the Clark/Wilson requirements can be found by examining the semantics of the classes assigned to the various data objects in the context of the basic non-discretionary and program integrity rules. The properties listed below are global, persistent, and unconditionally maintained by the underlying security kernel, provided the security officer and certifying officials are diligent in assigning clearances and assigning labels to certified transactions correctly.

An individual CDI has the "general" CDI integrity and disclosure category, an individual CDI category, and an individual CDI disclosure category. It, therefore, can be

manipulated using only programs that have been certified to correctly manipulate that specific CDI, by personnel authorized to both read and write CDIs in general and this CDI in particular.

An individual TP has the "general" TP and CDI integrity and disclosure categories, the integrity category of each CDI it has been certified to manipulate correctly, and an individual TP disclosure category. It, therefore, (as an executable object) can be used to manipulate any of the CDIs it has been certified for, and none of the others (it does not carry their integrity category, so program integrity will prevent it from being executed.) It may be created, modified, or deleted only by someone cleared "[t]" (i.e., a certifier). It can be read (and therefore, executed) only by someone cleared {t, TPi}: i.e., a TP user who has been specifically cleared to use it.

UDIs, for convenience, have been "bundled" into a single class [u]{u}. This class is intended to include uncertified programs. They cannot be used to manipulate CDIs or TPs (they do not have [c] or [d] integrity) and if someone is cleared only [u]{u}, (i.e., a programmer) that person cannot read or write either CDIs or TPs. Similarly, if certifiers are not cleared [u], certifiers cannot create or modify programs.

Finally, we note that if certifiers are not cleared {CDIK}, they cannot execute TPs against CDIs, in the sense that the TP will work: because their subjects will not be able to observe the state of the CDI. A certifier can always, by definition, certify a TP that will blindly modify a CDI (no matter who executes it): it would, therefore, be inconsistent to require that the system prevent this possibility.

Thus, the basic non-discretionary policy encompasses a good deal of what Clark/Wilson require, with a high degree of assurance, simply by issuing clearances and labeling certified transactions appropriately. The residual requirements are supporting in nature, and involve two major components: first, enforcement of separation of duties relative to certifiers and "TP users", and secondly, enforcement of Rule E2 (UserId differentiated restrictions on which transactions a user may use to manipulate a particular CDI, over and above those inherent in the set of CDIs the TP has been certified for). These components are intrinsically discretionary in nature (that is, they cannot be expressed in terms of information flow properties that are global and persistent).

Assignment of Access Classes to Subjects

This section provides a bridge between the non-discretionary and discretionary components of the design: it is a

specification of the intended use of the classes for labeling the subjects that will comprise the active component of the abstract system. The discretionary enforcement mechanism may be viewed as restricting those subjects that can be activated by a user, to those the user is authorized to activate by Rule E2 and others.

An ordinary TP subject is a subject, executing a TP (say, TPi) on behalf of some user authorized to execute a TP against some particular set of CDIs. The authorized set, of course, must be a subset of the CDIs that the TP is certified to manipulate properly. In order to read the required CDIs and to read (and execute) the TP, its readclass must include:

{t, c, TPi, CDIa, CDIb, . . .}.

In order to modify the required list of CDIs, its writeclass must include:

[t, c, CDIa, CDIb, . . .].

Putting this together:

Ordinary TP subject:
reads: CDIa, CDIb, . . .
CDIb . . .
writes: CDIa, CDIb, . . .
executes: TPi
subject writeclass:
[t, c, CDIa, CDIb, . . .]{}
subject readclass:
[]{t, c, TPi, CDIa, CDIb, . . .}.

We can now verify that the execution of this subject is correctly and completely constrained to what we want:

1. It can execute only TPi. It can read (and therefore) execute TPi under the basic non-discretionary policy because its readclass dominates the class of TPI. It can execute TPi under program integrity because its writeclass has less integrity than TPi. Because its writeclass includes [t], it can execute, by program integrity, only TPs. Because only {TPi} is included in its readclass, it can read (and execute) only one TP, namely TPi. It follows that the subject can execute TPi, and only TPi.

2. It can read any CDI included in its readclass, and no others. (It cannot read any others, CDIx for instance, because their disclosure categories are not in its readclass).

3. It can write any CDI included in its writeclass, and no others. (In confirming that it can write CDIa, for instance, the reader must remember that integrity dominance is inverted: the writeclass of the subject has a superset of the required integrity categories, and therefore, is dominated by the class of CDIa, because it has greater integrity).

4. It cannot read, write, or execute any UDI. (To read a UDI, it would need disclosure category {u} in its readclass, which it does not have. To execute a UDI, it must be able to read it. To write a UDI it would have to have [u] integrity.)

5. It can read, but not modify, any information in TPi, and can neither read nor write any other TP. It cannot modify any TP because no integrity category of the form [TPn] occurs in its writeclass. It can read TPi, but no other TP, because only {TPi} occurs in its readclass.

We must hasten to note that all TP subjects are trusted. This is probably the most disconcerting notion advanced in this paper, so it is important to understand why an appropriate basis for trust exists. The TP subject is trusted with regard to both integrity and disclosure, within the limits established by its readclass and writeclass. It is not trusted to write any but its assigned CDIs, or to read any but its assigned CDIs and TP. In particular, even though the TP program might be capable (as indicated by its access class) of reading or writing other CDIs not authorized to this subject, there is high assurance that for this subject, it cannot do so. To establish an adequate basis for the required trust, all that remains to be shown is that the TP program manipulates the CDIs in a way consistent with whatever it means for the CDIs to be collectively "correct": this, however, is defined by the certifier of the TP and, by assumption, the TP has been certified to be correct. We conclude that an adequate basis exists for trusting this subject.

Arguing in the same sort of way, we arrive at appropriate readclasses and writeclasses for special TP subjects (those executing TPs certified under rule C5 to manipulate UDIs as well as CDIs:

Special TP subject

reads:	UDIs, CDIa, ...
executes:	TPj
writes:	UDIs, CDIa, ...
writeclass:	[t, c, u, CDIa, ...]
readclass:	{TPk, t, c, u, CDIa, . . .}

The analysis of what this subject can do is similar to that of an ordinary TP subject.

Creation and Certification of TPs

We may also appropriately discuss at this point the installation of new TPs. Creation of a new TP is viewed as having three phases, with certain requirements involving separation of duties:

1. A programmer creates a UDI. We assume that it is undesirable for programmers to be able to even view CDIs, so that the creation of a malicious program tailored to manipulate selected CDIs would generally require collusion with a TP user.

2. A certifier selects the UDI and copies it to a certification workspace. It is required that the ability to copy UDIs, or modify candidate TPs in this workspace be confined to certifiers, and involve evaluated TCB code. (Generally, certifiers are allowed only to cause the copying of a UDI and the use of tools that assist in the certification process. They must not be able to either create new UDIs, or modify candidate TPs, or view or modify CDIs).

3. Once a candidate UDI in the certification workspace has been certified, certifiers must be able to install the candidate TP by copying it into a repository with the appropriate access class, as specified earlier. Note that because the Clark/Wilson "relations" are encoded in the access class of an installed TP, Rule E4 specifically precludes certifiers from being able to execute TPs, once installed.

It is convenient to take, as the access class for objects in the read-only certification workspace [cert, cert1]{cert}. We also provide for a certifiers read-write workspace of access class [cert]{cert}.

A programmers' subject is untrusted, and has readclass and writeclass of [u]{u}. It is assumed that a suite of programming tool, sample databases, productivity tools, and so on exists, all with access class [u]{u}. These are the only objects that can be read from or written to by a programmers' subject, as all other objects have additional disclosure and integrity categories.

programmers' subject:

reads:	UDIs
executes:	UDIs
writes:	UDIs
readclass:	[u]{u}
writeclass:	[u]{u}

A special subject can be activated by a certifier to transfer a candidate TP from the programmers' workspace to the certifiers' read-only workspace (integrity [cert, cert1]). It is assumed that there is a TCB trusted subject providing this service to appropriately cleared personnel (certifiers) with the following access classes:

```
subject to stage candidate TP:
     reads:          UDIs
     writes:     candidate TP with class [cert, cert1]{cert}
     executes:   TCB program with class [cert, cert1]{cert}
     readclass:  []{u, cert, cert1}
     writeclass: [cert, cert1]{}
```

The subject is trusted to correctly copy the UDI designated by its user (a certifier) into the certifiers read-only workspace. This is logically an integrity upgrade operation.

A certifiers' subject is allowed to read candidate TPs in the workspace, but not modify them. It uses read-only tools with integrity [cert, cert1], and a scratch workspace of integrity [cert] so that intermediate results cannot be mixed with the read-only candidate TPs. It is untrusted.

```
certifiers' subject:
     reads:      candidate TP with class [cert, cert1]{cert}
                 scratch data with class [cert]{cert}
     writes:     [cert]{cert}
     executes:   Certifiers tools with class
                 [cert, cert1]{cert}
     readclass:  [cert]{cert}
     writeclass: [cert]{cert}
```

A TP installation subject is used by a certifier to move (copy and delete) a candidate TP from the certification read-only workspace to the designated TP repository. By designating the particular access class for the finished TP, the certifier in effect is specifying the Clark/Wilson relation for the TP mentioned in Rule C2.

```
subject to install TP:
     reads:          [cert, cert1, t, c, all TPs, all CDIs]
                     {cert, t, c, all TPs}
     writes:         [cert, cert1, t, c, all TPs, all CDIs]
                     {cert, t, c, all TPs}
     executes:       TCB program with class
                     [cert, cert1]{cert}
     readclass:      []{cert, t, c, all TPs}
     writeclass:     [cert, cert1, t, c, all CDIs, all TPs]
```

This subject is highly trusted, as it must be able to move the TP into a repository labeled with any combination of CDI

integrity categories (as designated by the certifier), remove the TP from the read-only workspace. However, the program executed has integrity [cert1] and cannot be modified by either programmers, TP users, or certifiers.

Assignment of Clearances to Personnel

Having discussed the assignment of access classes to both subjects and objects, it is time to discuss the assignment of clearances to users. A clearance relates each UserId known to the system to the maximal sets of integrity and disclosure categories the user needs, and is trusted, to modify and/or view.

We will consider three general classes of users:

Ordinary TP users are allowed to observe and modify CDIs using TPs. Typically, the collection of CDIs the user is cleared to manipulate is smaller than that for the TPs for which the user is authorized. TP users will not be allowed to modify UDIs, although they may view UDIs (e.g., using special TPs). They also may not modify TPs or access the certifiers' workspace in any way.

An ordinary TP user has the following clearances:

integrity: [c, list of CDIs]
disclosure: {t, c, u (optional), list of CDIs}

It is easy to check that these clearances allow the user to activate a TP subject, given that the subject's CDis are consistent with those for which the user is cleared and that the user is cleared to read (and thus execute) the TP. It is also apparent that a TP user cannot activate a programmers subject or any of the certifiers subjects, as the requisite clearances are missing.

A programmer is allowed to create, view, and modify UDIs but nothing else. This corresponds to the following clearance:

integrity: [u]
disclosure: {u}

A check shows that a programmer will not be able to activate a TP subject or any certifiers subject based on clearance alone.

A certifier is allowed to activate any of the certifiers subjects, but may never view or modify a CDI. The appropriate clearances are:

integrity: [cert, cert1, t, c, all TPs, all CDIs]
disclosure: {cert, t, c}

Of some interest is the fact that a certifier could (in theory) modify a CDI, but can never read one, as the certifier does not possess a disclosure clearance for any CDI. This is semantically consistent: a certifier could (equivalently) certify a CDI that erases or inserts junk data into a CDI, but is unable to execute a TP that can read a CDI, and so modify it in an orderly way.

It is easy to imagine a tool in the TCB module providing the interface for the security officer, that would restrict the set of "grantable" clearances to those defined above and ensure that the list of CDIs in a TP user's clearance were consistent with the CDIs contained in the TPs access class. Such a module would also easily enforce separation of duties between certifiers, TP users, and programmers.

Rule E2

Use of the clearances above are insufficient to achieve the functionality implied by Clark and Wilson's Rule E2. This Rule mandates that the TCB maintain a list of triples of the form

(UserId, TPi, (CDIa, CDIb, . . .))

The intended meaning of each triple is that the particular user is authorized to execute TPi against the listed CDIs. If TP subjects were activated freely on behalf of users constrained only by the clearances, the user might be allowed to execute a given TP against CDIs included in the user's clearance because the user was authorized to perform a different transaction against them. When presenting the paper, Clark made a strong representation that the finer granularity of control is required: one ought to be able to authorize a user to execute TP1 against CDI1, and TP2 against CDI2, without authorizing TP1 against CDI2 as a result. However, the clearances specified above are not capable of providing this granularity of control.

The enforcement can be achieved by maintaining, within the TCB, the list of triples defined in Rule E2 as a protected database. During the logon procedure, the user, after being identified and authenticated, causes activation of a subject (or subjects) that execute on the user's behalf. At this time, the list of triples would be checked: only if the user had requested a subject consistent with a triple appearing in the database would the requested subject be activated. It is assumed that a trusted interface is provided (available only to those individuals authorized to maintain the triples database) for its maintenance. Clark and Wilson imply that it is a certification function to update it, although there is no reason this must be so, as long as the TCB clearance function ensures that the individuals authorized to maintain the triples do not have a TP user clearance.

Discretionary and Supporting Policy

We may now characterize the discretionary and supporting policy components of the Clark/Wilson requirements. The audit and identification/authentication requirements would seem to be straightforward, easily met by a conventional TCB. The previous section regarding maintenance and application of the Rule E2 "triples" have been characterized as intrinsically discretionary in nature. By this is meant "not non-discretionary": that is, the policy enforced is local and dynamic in nature, not reflecting global and persistant constraints on information flow. The term "discretionary" should not be equated with "enforced weakly" or "at the (complete) discretion of the user": it seems clear that the abstract design above enforces rule E2 with all of the force and assurance desired by Clark and Wilson. What has been changed from many discretionary systems (e.g., MULTICS) is the policy for control of the discretionary database. MULTICs has been described as incorporating a "laissez-faire" control policy: the system desired by Clark/Wilson is more centralized. The system of triples shares many intrinsic characteristics with other discretionary systems: in particular, one can (by examining the list of triples) determine exactly who can execute which TPs against which CDIs. This knowledge does not allow any inferences to be made about what a user can really do, as the effect of executing arbitrary sequences of permitted transactions is not apparent (nor can it be made apparent in any computable way -- the problem is intrinsic). This situation can be fruitfully compared with the non-discretionary policy: examination of a user's clearance shows exactly which information can be viewed or modified by that user: moreover, the controls implied can be bypassed only by use of trusted subjects, which (by definition, and in the system described) are shown (either by the evaluator, for TCB subjects, or by the certifier, for TPs) to preserve the integrity of CDIs. Thus, the global property of the system is that "CDI integrity is preserved": the discretionary property that "only authorized TP/CDI sets are executed by a particular user" is true with just as high a degree of assurance, but does not contribute to the intrinsic integrity of the CDIs.

Conclusions

The abstract design for a system implementing the Clark/Wilson requirements has now been presented. Sufficient detail has been provided so that it can be checked for consistency and correctness. It seems appropriate to conclude with some observations:

1. Although, in this case, the design serves as an "existence proof" that the requirements can be viewed as a combination of non-discretionary and discretionary policy components, it should be understood that it was virtually certain (supposing the requirements to be internally consistent)

that a decomposition into "non-discretionary" and "everything else" components could be found from the beginning. Denning in [7] shows how to reduce arbitrary information flow policies to lattices: "everything else" can always be provided as a combination of trusted subjects moving information "the wrong way" in accordance with some specialized disclosure downgrade and/or integrity upgrade local policy, plus further access control restrictions (e.g., based on additional object/subject attributes).

2. The primary intent in detailing this design was to understand how to build a system compliant with the requirements on an available TCB (GEMSOS): it is believed that this goal has been met.

3. A secondary result of the design is that it shows how to express "strong type enforcement" (for that is, in part, what the requirements demand) in terms of the well-understood semantics for conventional non-discretionary policies. The combination of isolated protection domains with programs certified to move data correctly from one domain to another, executable only by trusted subjects, coprresponds, it would appear, precisely with the notion of "typed domains" and "assured pipelines" advanced by Boebert and others [8]. This work may be of help in matching these ideas to currently established evaluation practique.

4. Another secondary result that might be useful is that it serves as a non-trivial example of the utilization and applicability of strict and program integrity, in a context that helps clarify the semantics implied by these policies.

References

[1] D.D. Clark and D.R. Wilson, A Comparison of Commercial and Military/Computer Security Policies. In Proc. 1987 Symp. on Security and Privacy. IEEE, April 1987.

[2] D.E. Bell and L.J.La Padula, Secure Computer Systems: Unified Exposition and Multics Interpretation. EDS-TR-75-306, The MITRE Corp., Bedford, MA, March 1976.

[3] K.J. Biba, Integrity Considerations For Secure Computer Systems. USAF Electronic Systems Division, Bedford, MA, ESD-TR-76-372, April 1977.

[4] L.J. Shirley and R.R. Schell, Mechanism Sufficiency Validation by Assignment. In Proc. 1981 Symp. on Security and Privacy, pages 26-32, IEEE, April 1981.

[5] Department of Defense Trusted Computer Systems Evaluation Criteria, Dept. of Defense, National Computer Security Center, Dec. 1985. DOD 5200.28-STD.

[6] J.H. Saltzer and M.R. Schroeder, The Protection of Information in Computer Systems. In Proc. IEEE 63, 9 (September 1975), pp. 1278-1308.

[7] D.E. Denning, On the Derivation of Lattice Structured Information Flow Policies. CSD TR 180, Purdue University, March 1976.

[8] W.E. Boebert and R.Y. Kain, A Practical Alternative to Hierarchial Integrity Policies. In Proc. 8th National Computer Security Conference. October 1985.

A10 **A PRACTICAL ALTERNATIVE TO HIERARCHICAL INTEGRITY POLICIES** [1]

W.E. Boebert*
R.Y. Kain**

*Honeywell Secure Computing Technology Center
Minneapolis, MN

**University of Minnesota (Honeywell Consultant)
Minneapolis, MN

BACKGROUND

The Secure Ada Target

The Secure Ada Target (SAT) project is an effort to develop a machine that meets and exceeds the A1 level of the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC). An overview of the machine is given in Reference 1.

Enhanced Security Policies

The SAT system design meets the A1 requirements with respect to the mandatory and discretionary policy requirements, and it exceeds the A1 level by enforcing an enhanced mandatory policy whose aim is to prevent corruption of sensitive information. Early versions of the machine incorporated a variant of the "traditional" hierarchical integrity policy; detailed analysis showed the inadequacy of this approach, and an alternative based on types and domains was developed.

PROBLEM STATEMENT

TCSEC Requirements

The TCSEC requires that systems at the B2 level of assurance and above demonstrate conformance to a security policy. The TCSEC further gives a set of minimum requirements that an acceptable policy must meet. Briefly stated, these requirements are that information be labelled internally with a security level and that accesses made by active subjects to information-holding objects be restricted in a manner that prevents information from flowing down in security level. We shall refer to this policy as the "compromise policy" and the security level used in its policy decisions as the "compromise level" of objects and subjects.

The TCSEC is silent on the equally important topic of preventing the corruption of sensitive information. A modular implementation of the TCSEC requirements dictates that it is necessary to impose proven constraints on information flow other than those imposed by the mandatory policy. This implication arises because the TCSEC requires that exported information be properly labelled with its compromise level. A modular implementation of this exportation process would have separate modules for label insertion and device control.

Practical secure systems also require constraints on information flow in order to defend against so-called "virus" attacks, to demonstrate assured data flow through cryptographic devices, and to enforce sophisticated security policies whose aim is to prevent aggregation and inference.

1 - This paper appears in the Proceedings of the Eighth National Computer Security Conference, Sept. 30 - Oct. 3, 1985.

First Efforts

An early response to the problems of information corruption was the development of "Integrity Policies," several variations of which are described in Reference 2. In effect, these policies add a second attribute to information (integrity level) and impose access restrictions in order to protect sensitive information from unauthorized modification.

INTEGRITY POLICIES

Varying Integrity Levels

The policies described in Reference 2 fall into two broad classes. In the first class, the integrity levels associated with subjects and objects may change. This class includes the Low-Water Mark Policy for Subjects and the Low-Water Mark Policy for Objects.

In the Low-Water Mark Policy for Subjects, a subject may neither modify objects nor send messages to a subject whose integrity level is greater than the one the sender currently has. The current integrity level of a subject is equal to the lowest integrity level of any object to which it has been granted observe access; hence the name "Low-Water Mark." "Execute" access is treated as a form of observe.

The Low-Water Mark Policy for Objects does not impose any restrictions on the ability of subjects to modify objects. Instead, the current integrity level of an object is set to the lowest integrity level of any subject that has been granted "modify" access to that object.

Integrity policies in the above class have seen little, if any, practical use, owing to the difficulties of administrating them and the pathological states which they allow (such as a subject being denied access to objects it has created).

Fixed Integrity Levels

The second broad class of integrity policies includes the Ring Policy and the Strict Integrity Policy. In these policies, the integrity levels of both subjects and objects are fixed. Under the Ring Policy, a subject may obtain "observe" access to any object, but may not modify objects nor communicate with subjects of higher integrity. The Strict Integrity Policy is the full formal dual of the compromise policy defined in the TCSEC. It consists of a Simple Integrity Condition, which states that a subject cannot observe objects of lesser integrity; an Integrity *-Property, which states that a subject cannot modify objects of higher integrity; and an Invocation Property, which states that a subject may only send messages to subjects of higher integrity.

This second class of integrity policies has fewer intrinsic difficulties than the first, and variants have been implemented in reference monitors.

General Principles

Both classes of integrity policies represent varying interpretations of the same general principle: information should only flow "down" in integrity. In order to avoid excessive detail, we will offer our critique of, and alternative to, the general class of policies that adhere to this principle. We will call such policies "hierarchical integrity policies." This class includes all policies which assign an attribute called "integrity level" to information and which then impose rules to prevent (to one degree of assurance or another) information at high integrity levels from being corrupted by information of low integrity.

Integrity and Compromise

It is tempting to view hierarchical integrity policies as duals or complements of the compromise policy mandated by the TCSEC. While such a relationship can be shown to exist formally (especially in the case of the Strict Integrity Policy), the relationship does not exist in the broader sense of intent and application.

In particular, the nature of a compromise policy is that controls are imposed on programs based upon the context in which they execute, and not upon the degree of trust placed in the programs themselves. In particular, a compromise policy such as that mandated by the TCSEC can be shown to prevent the compromise of information even if the programs being executed are hostile in their intent.

Such immunity from hostile programs cannot be obtained by using integrity policies. If there were a hostile program in the system, it could simply wait until it was executing in the context of a high-integrity subject and then work its damage on high-integrity information. Under the Low-Water Mark Policies and the Strict Integrity Policy, this danger is prevented by assigning integrity levels to programs and equating "observe" and "execute" access. In these policies, a high-integrity subject is therefore bound to executing high-integrity programs. In the Ring Policy, no such restriction exists, and the policy is trivially subvertible by Trojan Horse techniques.

From the above it can be seen that there is an essential difference between compromise and integrity: compromise level is more naturally bound to subjects and integrity level is more naturally bound to programs. Attempts to bind integrity level to subjects, as is done in the above policies, should lead to difficulties in application. We will show that such difficulties do in fact exist; they manifest themselves as an excessive need for the concept called "trust."

Trust

A "trusted subject" is one which is privileged to selectively violate the letter of a particular policy. The programs executed by the subject must be verified to ensure that the exception does not violate the intent of the policy. This in turn requires that the intent of the policy be explicitly stated; this is often no easy matter.

In the case of compromise policies, trusted subjects are those which are permitted to "write down," that is, to cause information to flow downward in compromise level. In the case of such subjects, the adherence to the "higher" policy is demonstrated by showing that the subject moves a trivial amount of information, that the movement of information is audited so that abuses can be detected, and/or that the movement takes place at the instigation of an authorized user (a so-called "downgrader").

If we follow the pattern of viewing integrity policies as the formal duals of compromise, then "integrity trust" is the privilege of "writing up" in integrity. As with compromise, we associate trust with "modify" access in order to simplify the discussion.

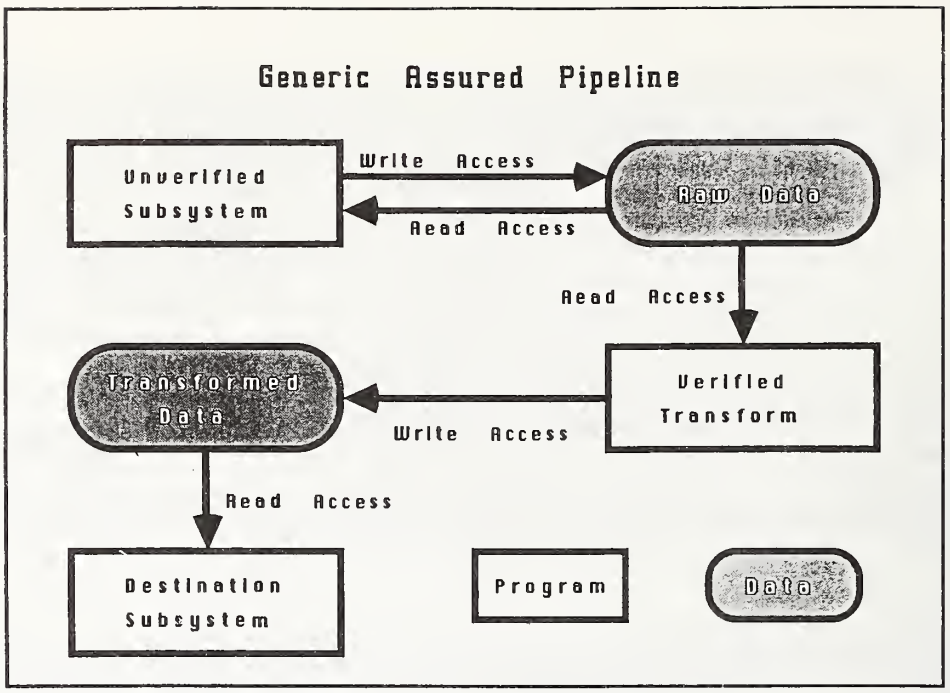
The attribute of trust, in the policies under discussion, is bound to subjects and not to programs. It is therefore necessary to prove that trust can never be abused; that is, that no hostile program can ever be executed within the context of a trusted subject. This in turn requires verification of usually complex low-level mechanisms that bind programs to subjects.

It is also necessary to state the intent of the policy being enforced and to formulate a subject-local property which captures that intent. It is then necessary to verify that the property is exhibited by all programs that could be executed in the context of the trusted subject. The use of trust therefore greatly complicates the proof process and reduces the degree of assurance in the system. It is accordingly a goal of the SAT effort to reduce the use of trust as much as possible, and it was this goal that led us to question and finally discard the notion of a hierarchical integrity policy.

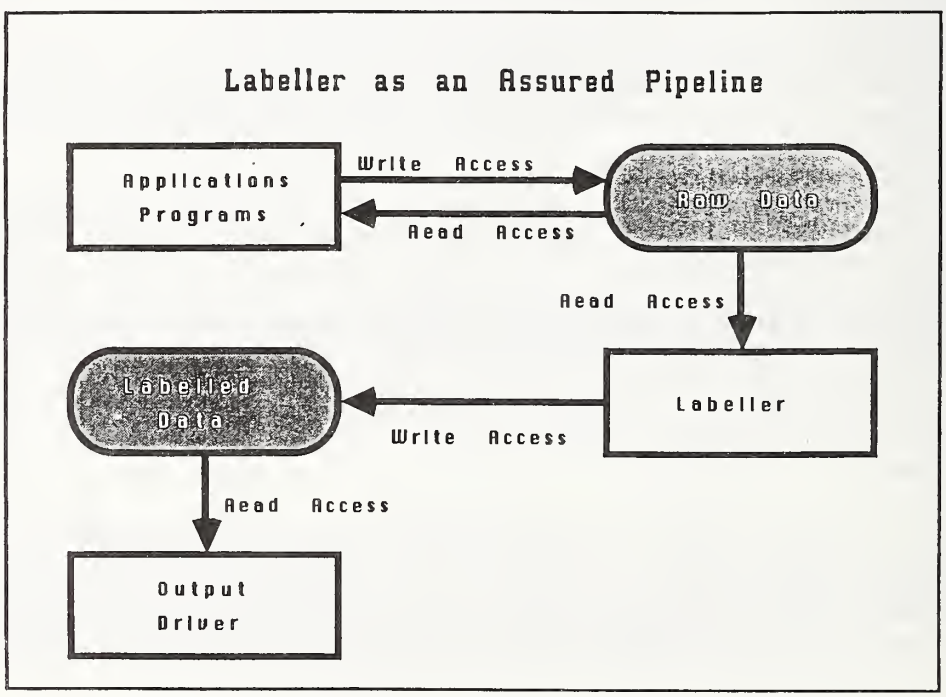
CRITIQUE

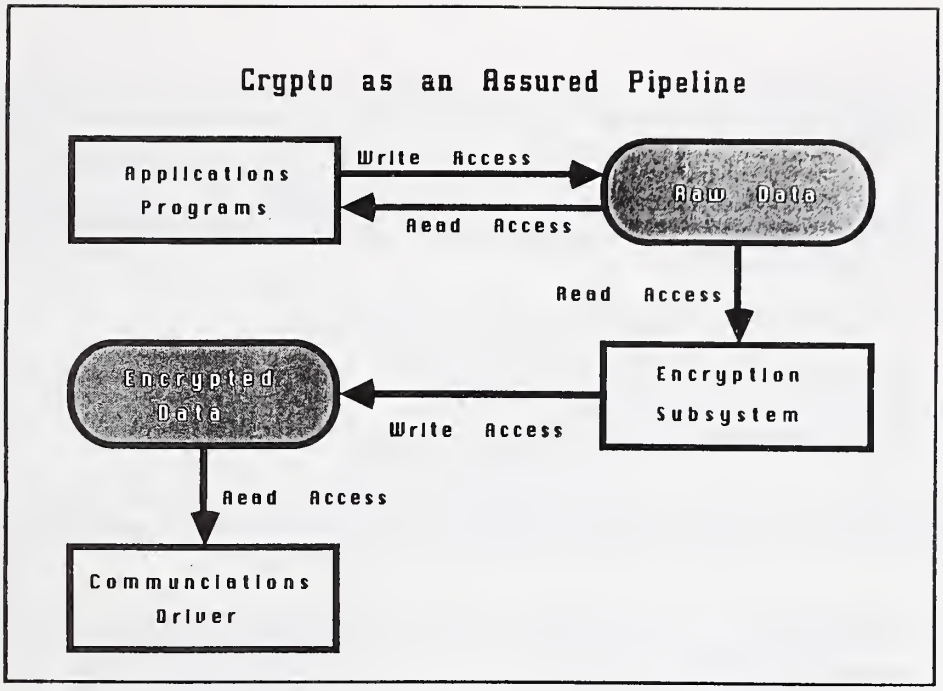
Assured Pipelines

In this section we will present a critique of hierarchical integrity policies. We will consider the shortcomings of such policies in the context of what we call an "assured pipeline;" a subsystem that is security-relevant and that must be encountered by data flowing from a particular source to a particular destination. Examples of assured pipelines are labellers and cryptographic subsystems. In Reference 3 we give an example of a similar subsystem that does not transform data, but instead selectively audits requests made to the reference monitor.



A labeller is a verified subsystem that converts the security level of an object from internal form to external form prior to the export of that object. The most common instance of a labeller is one that prints the classification level of a single-level object at the top and bottom of the pages when that object is output to a hard-copy device. A cryptographic subsystem encodes data in such a way that it may be safely downgraded and transmitted over an insecure communications path without effectively declassifying the information contained within that data.





From the above discussion, it can be seen that assured pipelines represent the most basic kind of structure that one would wish to construct and prove secure in a Trusted Computing Base.

Security of Assured Pipelines

To prove that an assured pipeline is secure requires the demonstration of three properties:

1. The transforming subsystem cannot be bypassed. That is, no hard copy can be printed without labels, and no information can go out on the insecure path in unencrypted form.

2. The transforms cannot be undone or modified once done. Data cannot be intercepted between labelling and printing and have the labels removed; data cannot be intercepted between encryption and transmission and have unencrypted information inserted.

3. The transforms must be correct. The labeller must insert external labels that are the proper representation of the internal label of the object; the cryptographic subsystem must properly implement the desired cryptographic algorithm.

The last property is the only property amenable to program proof techniques; the first two properties must be demonstrated by recourse to some global attribute of the underlying system. We will now show that enforcement of a hierarchical integrity policy is a poor candidate for such an attribute.

Integrity and Assured Pipelines

For simplicity, we shall use the labeller for hard-copy output in our discussion. Other labellers and cryptographic subsystems pose the same problems for hierarchical integrity policies; only the terminology used in the example will change.

There are two object types and two modules in this example of an assured pipeline. The object types are unlabelled and labelled data; the modules are the labeller and the output subsystem. Unlabelled data does not include the

printable classification levels at the top and bottom of pages; labelled data does. The labeller determines the security level of the object from its internal label, locates page boundaries, and inserts the proper label text. The output module is a device driver that causes the labelled data to appear on some appropriate hard-copy device.

The local security properties that must be proven of each of the modules are that the labeller selects the proper printable label and puts it in the proper place, and that the output module moves data to hard copy without modification to the label text.

The global security properties that must be proven of the pipeline are

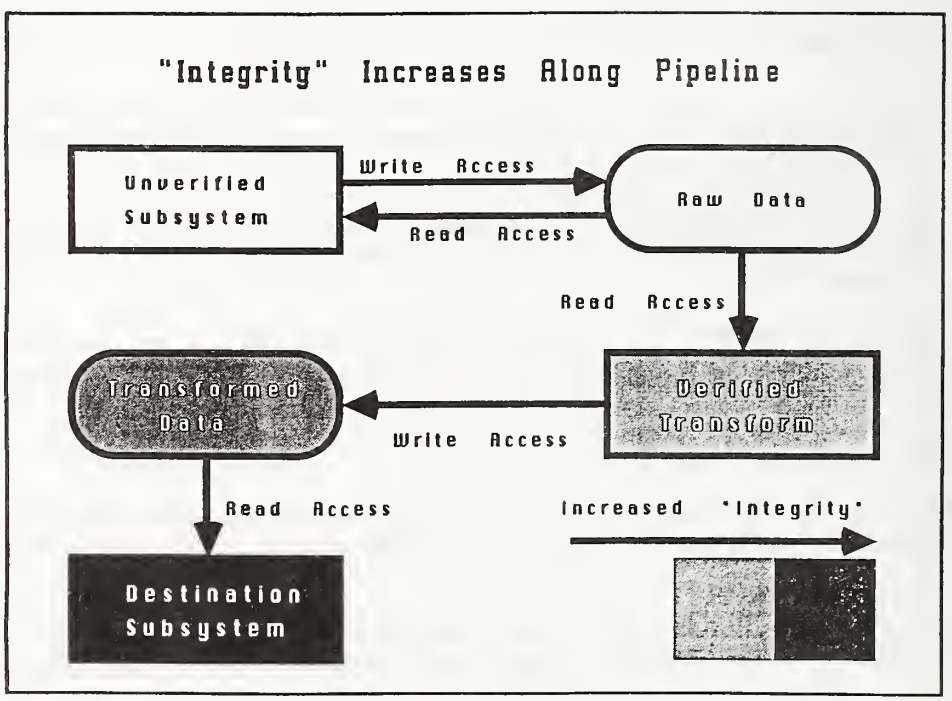
1. Only the labeller module produces labelled data.
2. Labelled data cannot be modified.
3. The output module will accept only labelled data.

We will now show that attempts to enforce these properties using a hierarchical integrity policy will inevitably involve the use of "trust" somewhere in the pipeline. Note that all information is at the same compromise level, so that the mandatory security policy imposed by the TCSEC is trivially satisfied.

There are three alternatives to assigning integrity levels in such a pipeline: the integrity levels of all data may be equal, the integrity levels may increase as data moves toward the output device, and the integrity levels may decrease as the data moves down the pipeline.

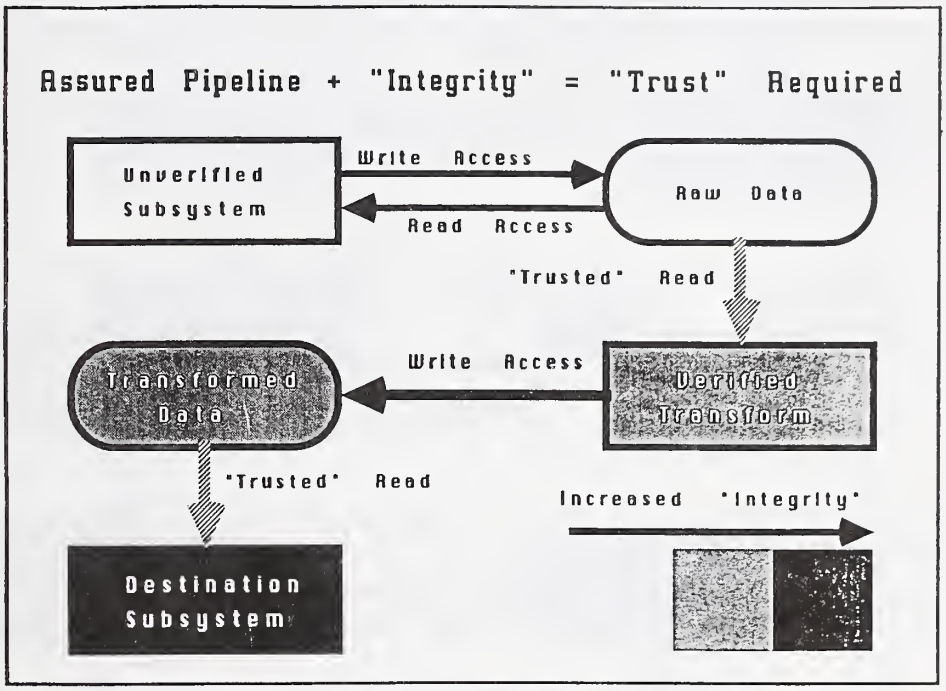
If labelled and unlabelled data are at the same integrity level, then no integrity policy will be able to distinguish between them. A hostile program will be able to remove or modify labels at will between the labelling and the output steps, and the output module will not be constrained by integrity level to outputting only labelled data.

If labelled data is at a higher integrity level than unlabelled data (the intuitive case), then trust must be invoked at each module in the pipeline, as it is clear that in such an arrangement information is flowing "up" in integrity.



The case where labelled data is at a lower integrity level than unlabelled has the same shortcomings as the equal integrity level case.

Thus, the application of hierarchical integrity policies to the most basic structure of a secure system either fails to enforce the desired restrictions or requires an exception from the policy at each step. We argue that this situation represents an excellent definition of the word "impractical," and offer an alternative that avoids these shortcomings and confers other benefits as well.



POLICY ENFORCEMENT IN THE SECURE ADA TARGET

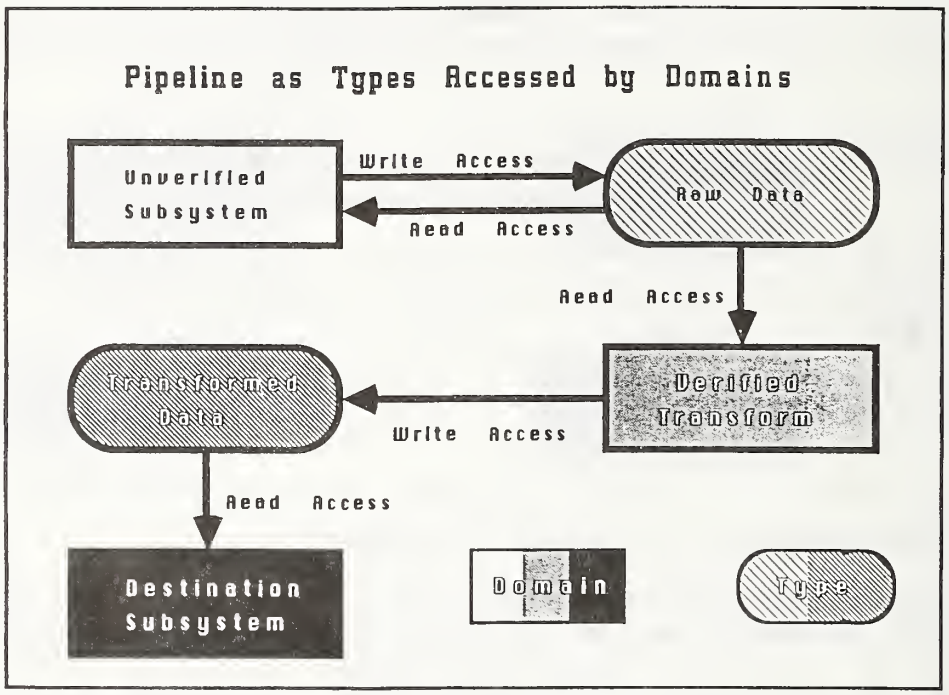
The SAT machine directly implements the reference monitor mandated by the TCSEC. The SAT reference monitor system checks every individual access attempt for consistency with the security policy being enforced by the system.

The SAT reference monitor is implemented in hardware and resides between the processor, which generates memory access requests, and the memory system, which satisfies these requests. The reference monitor intercepts illegal access attempts; an interrupt is caused when an illegal access is detected. For "normal" checking, the system aborts the offending subject, thereby guaranteeing that no illegal accesses can be completed and further that the program cannot obtain much information regarding the security state of the system by repeated attempts to make illegal accesses. (Otherwise, the system's security state might be used to construct a covert channel between two subjects.)

The SAT reference monitor is implemented by a combination of a memory management unit (MMU), which has conventional rights checking facilities, and a tagged object processor (TOP), a new module responsible for the system's protection state and the enforcement of that state. In particular, the TOP sets up the tables that define the access rights checked by the MMU. For system integrity, it is also necessary that the TOP be responsible for resource management and for the integrity of the internal state of the reference monitor. One important part of this state is the global object table (GOT), which contains a description of the security attributes of all objects within the system. In general, all elements of the system, including users, security properties, code, and data, are objects described within the GOT and managed by the TOP.

Of major concern are the security attributes of objects and their use in determining the access rights to be placed within the MMU during program execution. The basic SAT design starts with a minimum set of security attributes sufficient to satisfy both the mandatory and discretionary security policy requirements, which require comparisons between attributes of the subject in whose context a program is executing and attributes of the object to be accessed by that program. Thus security attributes are associated with both subjects and objects, and the TOP must make appropriate comparisons to establish proper access rights in the MMU.

Three security attributes are associated with subjects and three different attributes are associated with objects. Both subjects and objects have security (compromise) levels. Each subject is performing its function for some "user," whose identity is the second subject security attribute. The corresponding object attribute is its access control list (acl), which lists those users who are allowed access to the object's contents, along with the maximum access rights that each designated user is permitted. The third subject security attribute is the "domain" of its execution, which is an encoding of the subsystem of which the program is currently a part. The corresponding object security attribute is the "type" of the object, which is an encoding of the format of the information contained within the object.

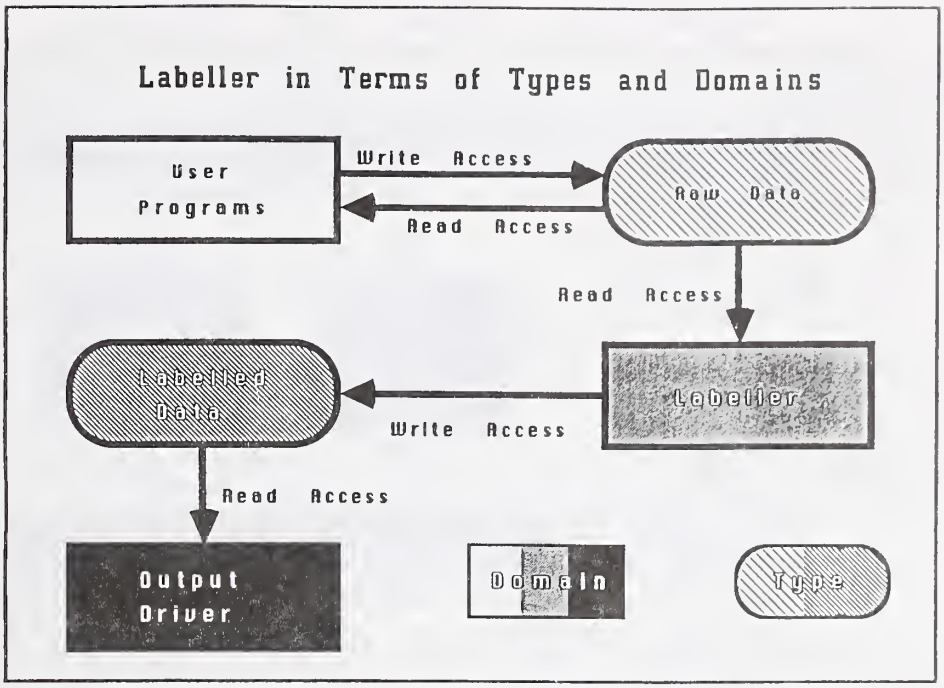


The process of determining the access rights to be accorded a particular subject for access to a particular object uses all of these three security attributes, as follows.

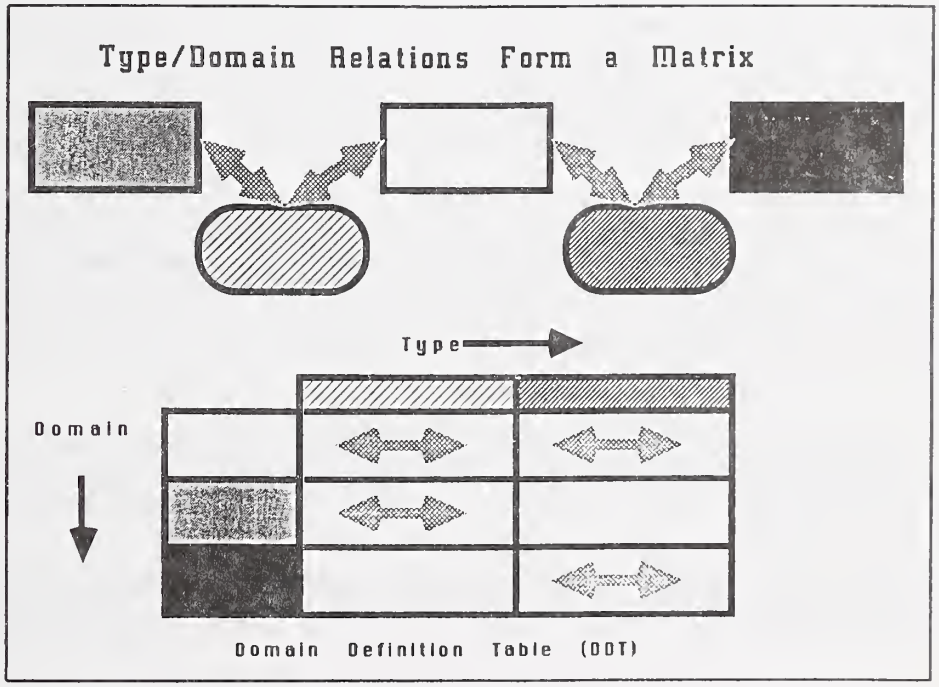
1. To enforce the mandatory access policy, the TOP compares security levels of the subject and of the object, and computes an initial set of access rights according to the algorithm defined in Section 4.1.1.4 of the TCSEC.

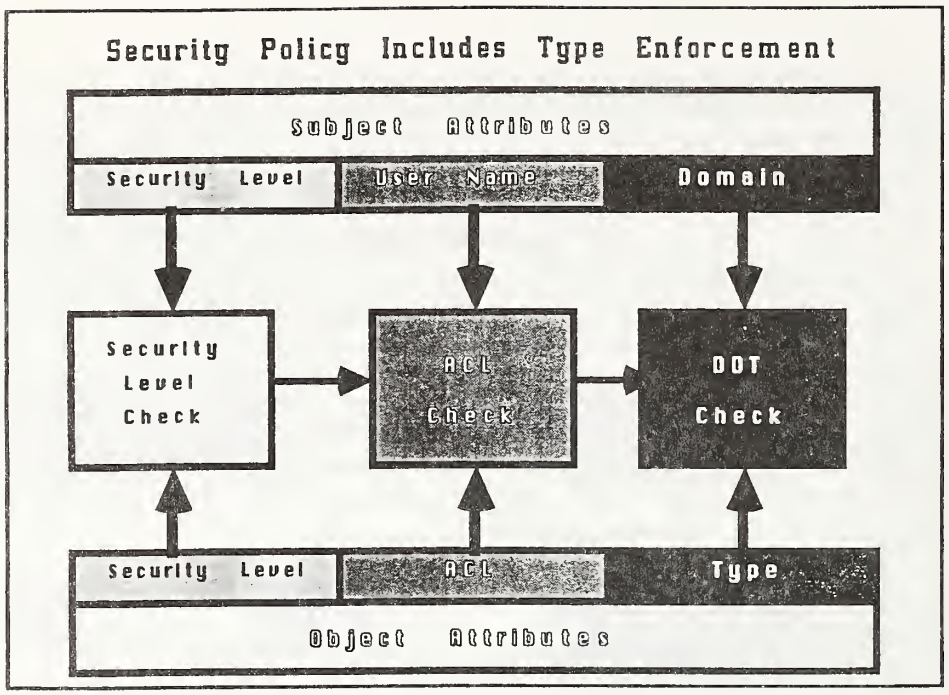
2. To enforce the discretionary access policy, the TOP checks the acl for the object; the acl entry that matches the user portion of the subject's context is compared against the initial set of access rights from the mandatory policy computation. Any access right in the initial set that does not appear in the acl is deleted from the set. The result is an intermediate set of access rights.

3. The third SAT access rights determination check compares the subject's domain against the object's type. Each domain is itself an object, and one of its attributes is a list of the object types accessible from the domain and the maximum access rights permitted from the domain to each type.



Conceptually the aggregation of these domain definition lists constitutes a table, which we call the Domain Definition Table (DDT). To make the domain-type check, the TOP consults the DDT row for the executing domain, finds the column for the object's type, and compares the resultant entry against the intermediate set of access rights. Any right in the intermediate set that does not appear in the DDT entry is dropped, and the result is the final set of access rights that is transmitted to the MMU.





(Certain domains have additional, privileged roles and may therefore obtain access rights in excess of those determined from the mandatory and discretionary checks. A discussion of this mechanism is beyond the scope of this paper.)

The above complex process cannot be performed on every access attempt. On the other hand, the checks cannot be made far in advance and saved (in a "capability," for instance), as such early binding cannot provide the access right revocation implicit in certain acl changes.

In SAT, the TOP operation load name space table (LNST) evokes the access rights check; it inserts access to a designated object at a designated segment number in a subject's address space, and establishes the correct maximum access rights for that subject to that object. The mandatory, discretionary, and domain rights checks are performed during the execution of LNST, and then the subject's MMU table is modified to reflect the new entry. If the LNST operation is proved to conform to the security policy and if the MMU is proved to enforce the access rights set in the NST, the system is thereby proved to conform to the security policy for each and every instruction execution.

Domain changing may occur as a side effect of procedure call. If the called procedure is not executable within the caller's domain, either the call is illegal or a domain change is necessary to complete the call. Information concerning domain changes is stored in a Domain Transition Table (DTT), which is stored as a set of lists associated with the calling domain. The SAT system creates new subjects to handle domain changes, as required. When a call requires a domain change, SAT suspends the calling subject and activates the called subject. The called subject has a different execution context, name space, and access rights, which will prevail for the duration of the procedure's execution.

In the SAT prototype, the DDT and DTT are set at the time that a particular version of the reference monitor is installed. The number of types and domains, and the relationship between them, accordingly remains static until a newer version of the reference monitor is installed. Later versions of SAT will include facilities for the dynamic creation of types and domains.

Note that the access right computation involves the successive denial, or "crossing off" of those access rights initially allowed by the mandatory policy. This approach guarantees that omission of an access right in a DDT

entry for a type, domain pair will effectively block access to that type by any program encapsulated in that domain. This guarantee is verifiable by inspection of the DDT and provides assurance that certain types remain "private" to certain domains. Note also that it is possible to assign types to procedure objects and place restrictions on "execute" access in the DDT. This last feature permits assurance that critical code is indeed encapsulated in protected domains. In effect, the DDT reflects, and gives assurance in, the structure of the reference monitor. This in turn permits a strong correspondence to exist between the organization of the design and the organization of the proof.

USES OF TYPE ENFORCEMENT

Implementing Integrity Policies

We would like to begin by observing that our type enforcement policy subsumes the second class of hierarchical integrity policies, that is, those in which an unchanged integrity level is bound to subjects and objects.

In order to implement a hierarchical integrity policy in SAT, it is necessary to first assign types to procedures based on their integrity level. The set of procedures possessing a given type is isolated into a distinct domain, which is the only domain from which these procedures may be executed.

Data objects are then assigned a distinct set of types, also based on integrity level. It is then trivial to devise a DDT configuration that implements the restrictions of the Ring Policy or the Strict Integrity Policy.

For example, let us assume that we have three integrity levels: 1, 2, and 3. We would then have three types of procedures: P1, P2, and P3 (with the corresponding domains), and three types of objects: O1, O2, and O3. It is also necessary to have a "gatekeeper" domain, P4, for use when integrity level changes are required.

In order to implement the Strict Integrity Policy, we need only construct a DDT configuration as follows.

Object Type:	O1	O2	O3
Domain P1:	o/m	o	o
Domain P2:	m	o/m	o
Domain P3:	m	m	o/m
Domain P4:	null	null	null

(o = observe; m = modify)

And the following DTT configuration:

Called Domain:	P1	P2	P3	P4
Domain P1:	e	e	e	cP4
Domain P2:	null	e	e	cP4
Domain P3:	null	null	e	cP4
Domain P4:	cP1	cP2	cP2	e

(e = execute and stay in current domain; cDestination = change to domain Destination.)

Tables for the Ring Policy may be similarly constructed. Note that a binding which is stated in the policy as existing between integrity levels and subjects is here mapped onto a binding between, in effect, integrity levels and procedures. This mapping is possible because the policy treats execute and observe access the same, thereby establishing a relationship between the integrity level of the subject and the integrity level of the procedure executing in the context of that subject.

The above argument shows that any set of restrictions enforceable by the second class of integrity policies is enforceable by the type enforcement policy. The first class of integrity policies, in which integrity levels of subjects or objects change, may be dismissed as impractical from the point of view of performance and proof.

Having argued that type enforcement can deal with any case that a hierarchical integrity policy can deal with, we proceed to the more interesting cases in which hierarchical integrity policies must appeal to "trust" in order to accommodate practical processing requirements.

Assured Pipelines

We will now show that the assured pipeline structure can be readily accommodated by the type enforcement policy. We will show DDT and DTT configurations based on the following types and domains:

- Types: unlabelled and labelled data
- Domains: user, labeller, and output

Unlabelled data is data that has only internal labels associated with it. Labelled data is data that is properly marked on the top and bottom of each page for output.

Unverified and possibly hostile programs are encapsulated in the User domain. The labeller module described in the previous section on assured pipelines is encapsulated in the Labeller domain and is verified to properly translate internal labels to readable form and place them in the correct positions in the data. The output module of the previous assured pipeline description is encapsulated in the Output domain and is verified to not tamper with labels. None of the domains in the example invoke any form of privilege.

The DDT that enforces the pipeline is as follows:

Object Type:	Unlabelled	Labelled
User Domain:	o/m	null
Labeller Domain:	o	o/m
Output Domain:	null	o

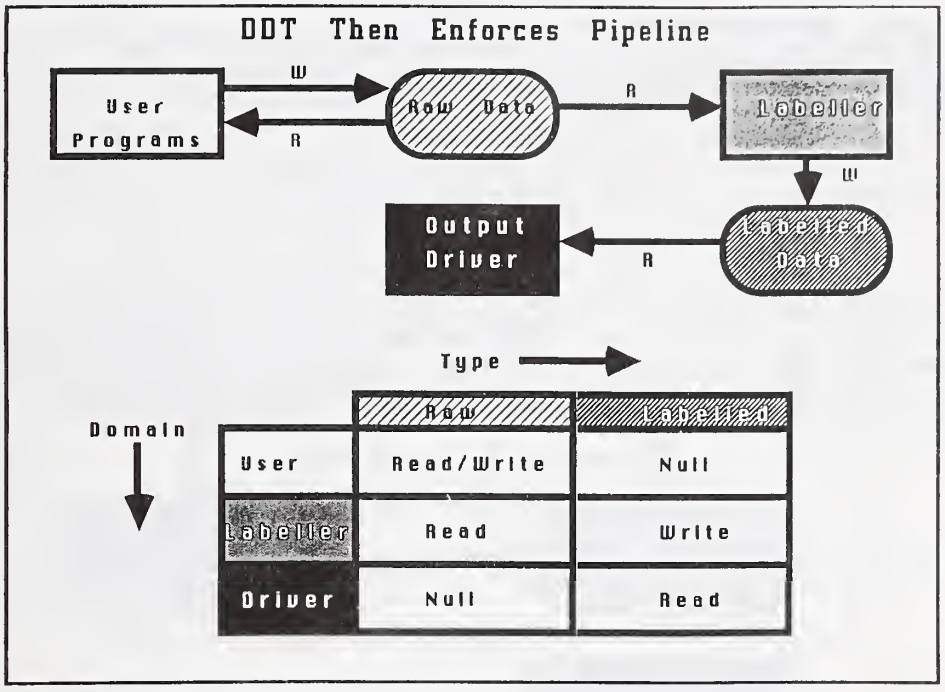
(o = observe; m = modify)

And the corresponding DTT is:

Called Domain:	User	Labeller	Output
User Domain:	e	cLabeller	null
Labeller Domain:	null	e	cOutput
Output Domain:	null	null	e

(e = execute and stay in same domain; cDestination = change to domain Destination.)

Note that not only does the DDT restrict the data flow, but the DTT restricts the control flow in such a manner that the pipeline must be initiated by (possibly hostile) user code in a proper manner; the Output domain is not callable from the User domain.



TYPE ENFORCEMENT AND PROOF

Factored Proofs

Assurance, in the final analysis, is based on human confidence; and confidence comes from insight and understanding. It has accordingly been a goal of the SAT project that its proofs of security be accessible to human analysis, understanding, and criticism.

This goal has led us to avoid the machine-generated proofs of previous efforts in favor of proofs that have an informally understandable underlying structure; formalism is used to permit machine-checking of our results and not as an end in itself.

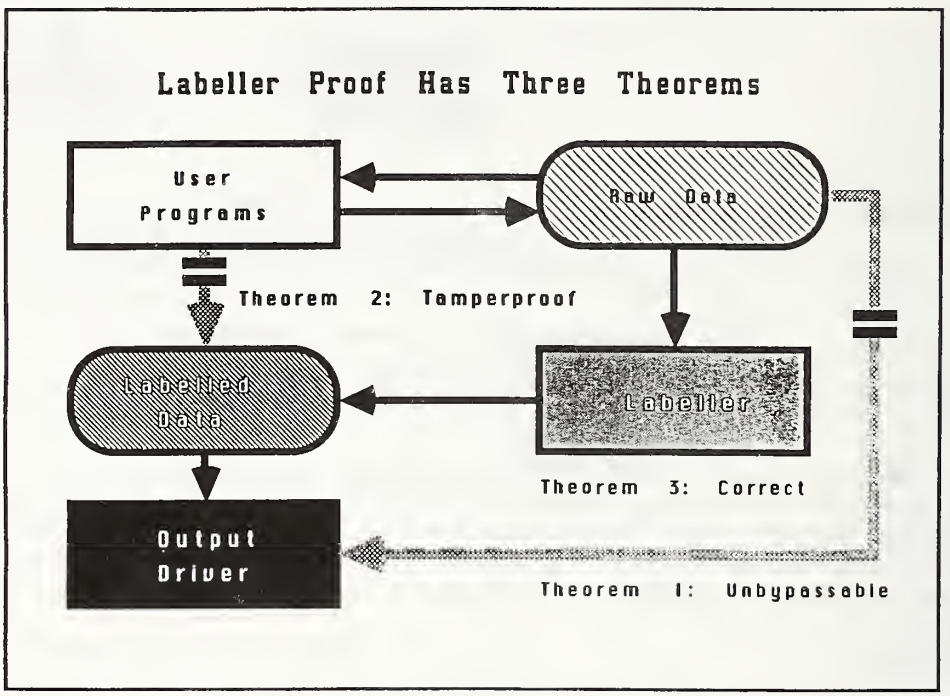
We use the traditional structure of a "factored" proof; that is, an argument based on an orderly presentation of lemmas. The proof has two purposes. The secondary purpose is to convince a skeptical observer that our system is secure; the primary purpose is to give that observer insight into the precise meaning we give to the word "secure."

In order to achieve this goal we must present a proof whose organization corresponds in a fairly obvious way with the organization of the system, so that for every conclusion we draw along the way there is a clearly identified system feature which supports that conclusion. In the next section we shall outline such a proof of our example labeller pipeline.

A Factored Proof of a Labeller

The fact that a labeller is "secure" can be captured in three theorems:

Theorem 1: Only labelled information is output to hard copy.
Theorem 2: Labels are properly inserted prior to output of labelled information.
Theorem 3: Labels are not modified prior to output of labelled information.



We now present the lemmas used in our proof, and the manner in which each lemma would itself be proven.

Lemma 1: The SAT hardware properly enforces a given DDT and DTT configuration. This lemma is proven as part of the overall proof of the security of the SAT reference monitor, and is accordingly "built in" to the SAT hardware.

Lemma 2: Only the Labeller module can write to Labelled data. This lemma is proven by inspection of the DDT configuration given in the example in the previous section.

Lemma 3: The Output module will read nothing but Labelled data. Again, this is proven by inspection of the same DDT configuration.

Lemma 4: The Labeller module properly translates internal labels to external form, and inserts them at the top and bottom of each page. This lemma is proven by applying standard program proof techniques to the labeller program. The proof involves demonstrating the truth of two relatively weak assertions: that the Labeller performs a table look-up properly and that it can find the top and bottom of a page of hard copy.

Lemma 5: The Output module does not tamper with labels. As a practical matter, this lemma will be proven using informal methods. This is because Output modules are typically complex and machine-dependent. It is

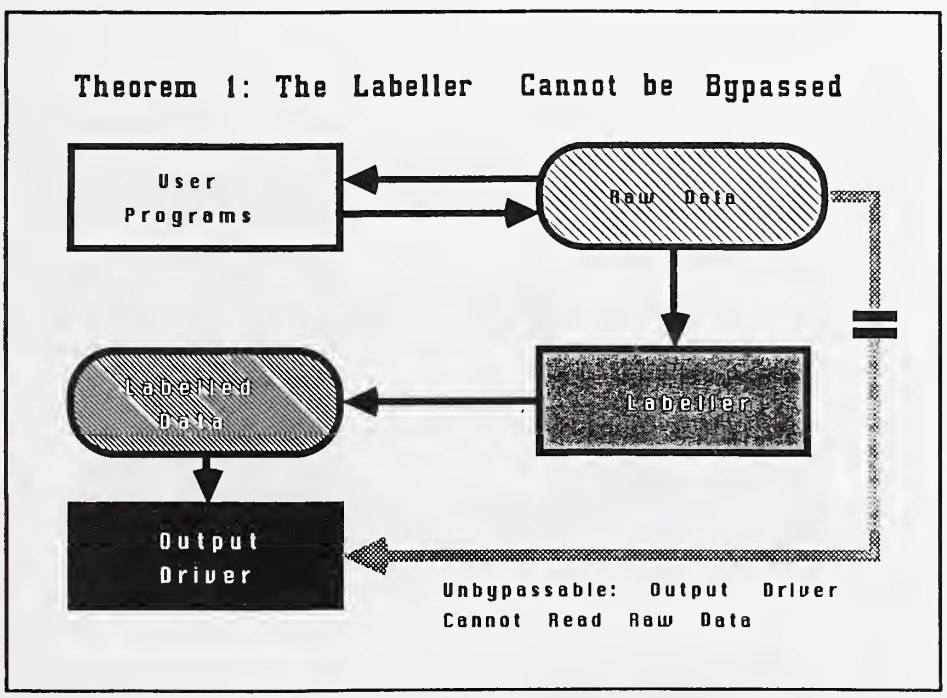
accordingly difficult to capture their operation in the semantics of formal program-proof systems. Modules of this type are amenable to inspection and comprehensive testing, especially when it is known (as in this case) that their inputs come only from formally verified code and therefore form a tractable set of test cases.

We now note the correspondence between this set of lemmas and the organization of the SAT reference monitor. Lemma 1 is a "hardware level" lemma, a global property that applies to all programs which execute on the SAT hardware, irrespective of their context or construction. Lemmas 2 and 3 are "structural" or "programming in-the-large" lemmas, properties which reflect the modular decomposition of the SAT reference monitor but which are not concerned with the internals of the modules themselves. Lemmas 4 and 5 are "programming in-the-small" lemmas, conclusions drawn about the operation of the modules that are independent of their context in the system. Thus we argue that there is a clear intuitive correspondence between elements of the system and elements of the proof.

Previous efforts to prove the security of labellers have generally been restricted to Lemma 4 and occasionally Lemma 5. That is, the proof has demonstrated that if the Labeller is invoked, then it properly labels; the proof does not demonstrate that the Labeller must always be invoked. In logical terms, the proof fails because a necessary but not a sufficient condition has been demonstrated; in design terms, the proof fails because the correctness of a module's internals has been shown, but the correctness of the structure of the system has not. This situation is analogous to proclaiming a system correct when its modules have all passed unit test, but integration testing has not yet been performed.

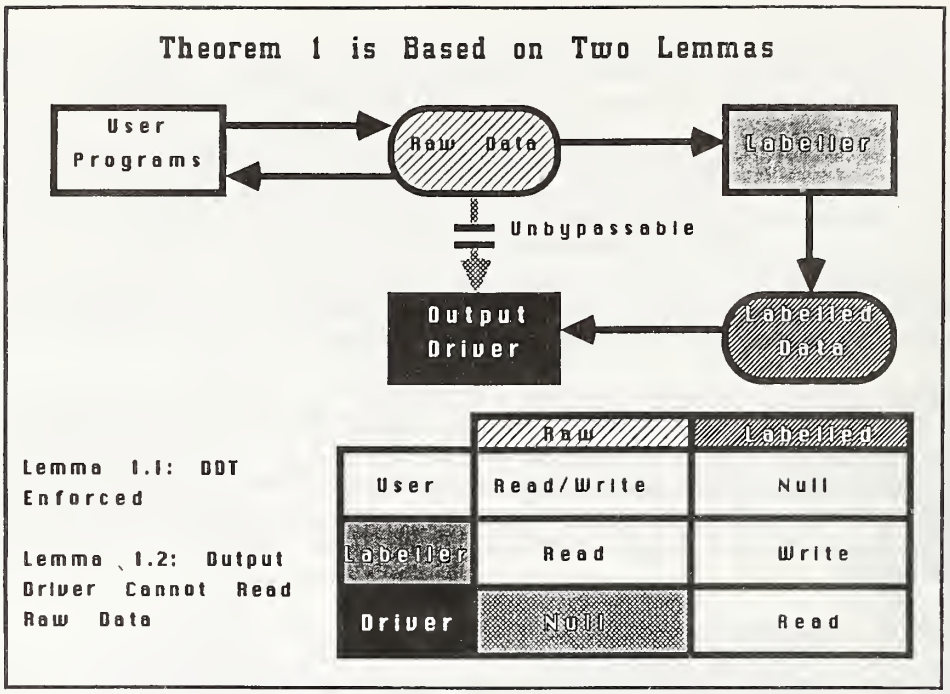
Given the above lemmas, the proof of each theorem is as follows:

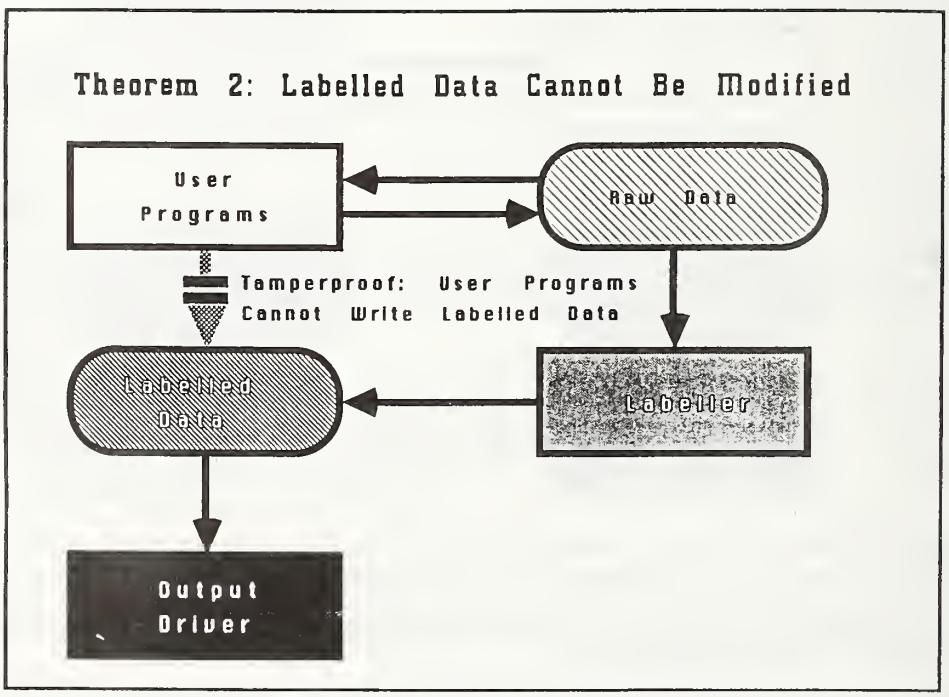
Theorem 1 (Only labelled data goes out): Lemma 1 (DDT enforced) and Lemma 2 (only Labeller writes Labelled) and Lemma 3 (Output only outputs Labelled).



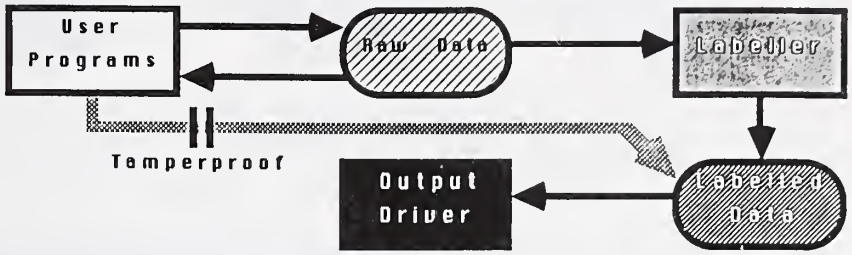
Theorem 2 (Labelled data is correct): Lemma 1 (DDT enforced) and Lemma 2 (only Labeller writes Labelled) and Lemma 4 (Labeller labels properly).

Theorem 3 (Labelled data is tamperproof): Lemma 1 (DDT enforced) and Lemma 2 (only Labeller writes Labelled) and Lemma 5 (Output module is benign).


Theorem 1 is Based on Two Lemmas
User Programs
Raw Data
Labeller
Unbypassable
Output Driver
Labelled Data
Lemma 1.1: DDT Enforced
Lemma 1.2: Output Driver Cannot Read Raw Data
Raw
Labelled
User
Read/Write
Null
Labeller
Read
Write
Driver
Null
Read


Theorem 2: Labelled Data Cannot Be Modified
User Programs
Raw Data
Tamperproof: User Programs Cannot Write Labelled Data
Labelled Data
Labeller
Output Driver

Theorem 2 is Based on Two Lemmas

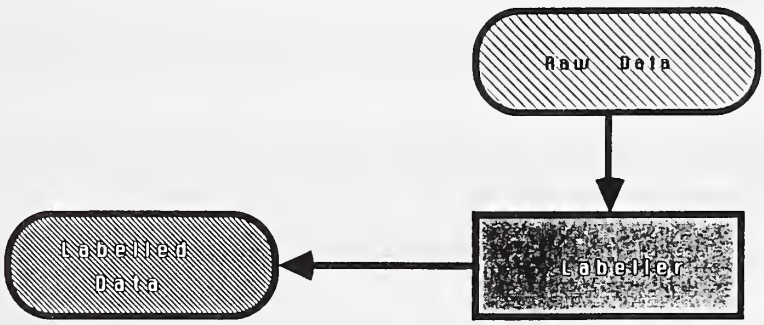


Lemma 2.1: DDT Enforced

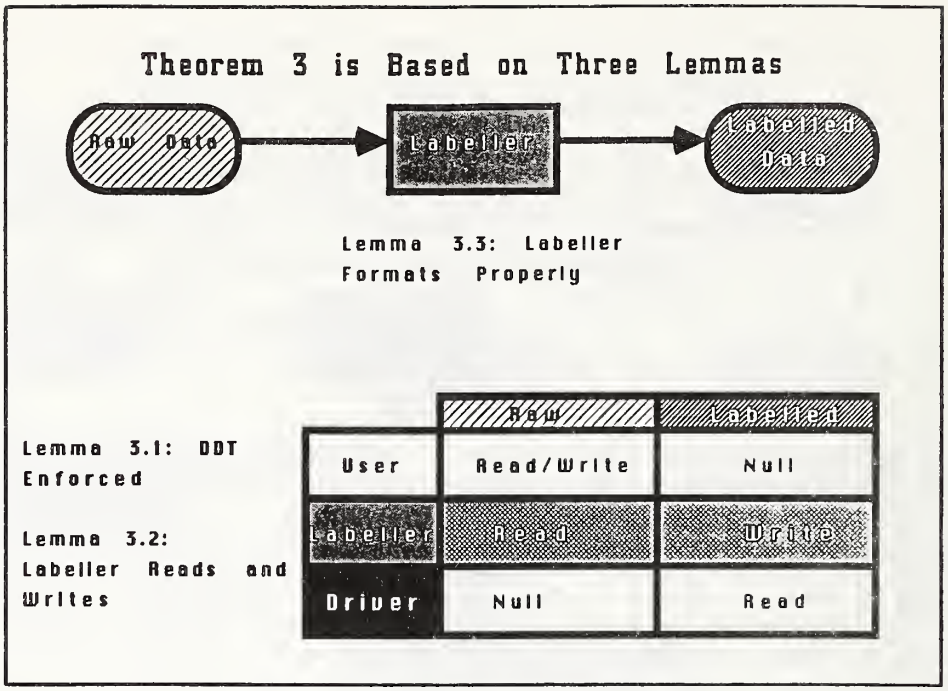
Lemma 2.2: User Programs Cannot Write Labelled Data

	Raw	Labelled
User	Read/Write	Null
Labeller	Read	Write
Driver	Null	Read

Theorem 3: Labeller is Correct



Labeller Program Implements Its Specification



SUMMARY

Hierarchical integrity policies have been shown to be inadequate to enforce the restrictions on information flow required by practical systems. An alternative policy based on types and domains has been presented that has been shown to subsume both the practical variations of hierarchical integrity policies and cases which such policies cannot handle without recourse to exceptions. The alternative is also shown to support proofs whose structure corresponds in obvious ways to the structure of the system being reasoned about.

ACKNOWLEDGMENTS

This effort has been supported by US Government Contracts MDA904-82-C-0444 and MDA904-84-C-6011.

REFERENCES

1. W.E. Boebert, R.Y. Kain, W.D. Young, and S.A. Hansohn, "Secure Ada Target: Issues, System Design, and Verification," *Symposium on Security and Privacy*, IEEE, 1985, 176-183.

2. K.J. Biba, "Integrity Considerations for Secure Computer Systems," The MITRE Corporation, Bedford MA, MTR-3153, 30 June 1975.

3. W.E. Boebert and C.T. Ferguson, "A Partial Solution to the Discretionary Trojan Horse Problem," *Proceedings of the 8th National Computer Security Conference.*

FURTHER REFERENCES (appeared after initial publication)

O.S. Saydjari, J.M. Beckman, and J.R. Leaman, "Locking Computers Securely," *Proceedings of the 10th National Computer Security Conference*, NBS, 1987.

W.D. Young, P.A. Telega, W.E. Boebert, and R.Y. Kain, "A Verified Labeller for the Secure Ada Target," *Proceedings of the 9th National Computer Security Conference*, NBS, 1986.

A11

Documents Available
from the
Workshop on Integrity Policy in Computer Information Systems
(WIPCIS)

Please circle the number of those items you wish to receive in the following list, and return a copy of it to Stephanie Sara-Pfeiffer, SRI International, Menlo Park, CA 94025.

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

2. Non-Discretionary Controls for Commercial Applications. Steven B. Lipner, Digital Equipment Corp.

3. Letter criticizing the Clark-Wilson Model from Robert H. Courtney, Jr., to Dr. Stuart W. Katzke. May 21, 1987. [Attachment 2]

4. Using Mandatory Integrity to Enforce "Commercial" Security. (Draft) T.M.P. Lee, Trusted Information Systems, Inc. October 26, 1987.

5. Locking Computers Securely. O. Tami Saydjari, Joseph M. Beckman, Jeffrey R. Lesman. NCSC.

6. A Practical Alternative to Hierarchical Integrity Policies. W.E. Boebert. Proceedings of the Eighth National Computer Security Conference, 9/30/85.

7. LOCK Implementation of the Clark-Wilson Rules. W.E. Boebert. WIPCIS. 10/27/87.

8. Data Integrity in a Business Data Processing System. W.H. Murray. 10/24/87.

9. Position Paper for Working Group on Granularity. W.H. Murray. WIPCIS. 10/27/87.

10. Outline for the Verification Working Group. W.E. Boebert. WIPCIS. 10/27/87.

11. Agreement with the External Environment. David R. Wilson. WIPCIS. 10/27/87.

12. Availability Issues, Final Report. Peter Wild. WIPCIS. 10/29/87.

13. Working Group on Granularity and Functions, Final Report. WIPCIS. 10/29/87.

14. Proposal to Extend the Scope of the Clark-Wilson Model to Include Availability. Donn B. Parker. WIPCIS. 10/29/87.

15. Clark and Wilson Commercial Integrity Model Slides. Donn B. Parker. SRI International.

16. List of Invitees to IEEE/ACM Invitational Workshop on Trusted Commercial Systems Criteria.

17. Participant Listing. WIPCIS. 10/29/87.

18. Framework for Secure Open Systems. Second Draft. European Computer Manufacturers Association. 8/87.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NIST/SP-500/160	2. Performing Organ. Report No.	3. Publication Date January 1989

4. TITLE AND SUBTITLE

Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)

5. AUTHOR(S)

Dr. Stuart Katzke and Zella Ruthberg, Editors

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)

ACM/Special Interest Group on Security, Audit and Control
IEEE/Computer Society, Technical Committee on Security and Privacy
National Computer Security Center

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 88-600606

☐ 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 is the Report of the Invitational Workshop on Integrity Policy in Computer Information Systems which was sponsored by the IEEE Computer Society's Technical Committee on Security and Privacy, the Special Interest Group on Security, Audit, and Control (SIGSAC) of the Association for Computing Machinery, the National Computer Security Center, and the Institute for Computer Sciences and Technology at the National Bureau of Standards. The workshop established a foundation for further progress in defining a model for information integrity. The workshop was held in response to the paper by David Clark of M.I.T. and David Wilson of Ernst and Whinney entitled "A Comparison of Military and Commercial Data Security Policy." The report's 10 sections contain an introduction, the composition of the organizing committee with a list of participants and a workshop agenda, a summary report by Donn Parker and Peter Neumann of SRI International, the reports of the five working groups, a response by Clark and Wilson, and a proposal by the National Bureau of Standards for continuing the effort to define an integrity policy. The appendices include a copy of the original Clark-Wilson paper, a summary of the Clark-Wilson rules, a number of position papers submitted in advance of the workshop, several papers submitted during and following the workshop, and a list of reference materials related to the integrity policy effort.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)
commercial computer system; computer information system; data security; discretionary controls; granularity; identity verification; integrity model; integrity policy; mandatory controls

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
195

15. Price

USCOMM-DC 6043-P80

☆U.S. GOVERNMENT PRINTING OFFICE:1989-242-311:92625

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & 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)

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