Borrower: IPL

Lending String: *IAI,JHA,JHA,MTE,MTE

Patron: Thomas, Jacques

Journal Title: Proceedings of the 8th National Computer Security Conference ; 30 September-3 October 1985, Gaithersburg, MD.

Volume: **Issue:**
Month/Year: October 1985**Pages:**

Article Author: National Computer Security Conference (8th ; 1985 ; National Bureau of Standards)

Article Title: WE Boebert, RY Kain; A Practical Alternative to Hierarchical Integrity Policies

Imprint: [Washington, D.C.? ; U.S. G.P.O.], 1985.

ILL Number: 12750853
12750853

Call #: **QA76.9.A25 N361985**

Location: **5**

ARIEL
Charge
Maxcost: 65.00IFM

10

Shipping Address:
Purdue University Libraries ILL
Purdue University
504 West State Street
West Lafayette IN 47907-2058

Fax:
Ariel: 128.210.125.135

Borrowing Notes; *Thank you.*

A PRACTICAL ALTERNATIVE TO HIERARCHICAL INTEGRITY POLICIES

W.E. Boebert

Honeywell Secure Computing Technology Center
Minneapolis MN

R.Y. Kain

University of Minnesota
(Consultant to Honeywell)
Minneapolis MN

BACKGROUND

The Secure Ada Target

The Secure Ada Target (SAT) project is an effort to develop a machine which 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 it 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 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 it is to prevent aggregation and inference.

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 which 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 "up" in integrity. In order to avoid excessive detail, we will offer our critique of, and alternative to, the general class of policies which 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 insure 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 which 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 which 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 which is security-relevant and which 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 which does not transform data, but instead selectively audits requests made to the reference monitor.

A labeller is a verified subsystem which 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 which 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 which 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 which 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 subsytem. 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 which causes the labelled data to appear on some appropriate hard-copy device.

The local security properties which 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 which 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 labelled data only.

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.

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.

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 which does not appear in the acl is deleted from the set. The result is an intermediate set of access rights.

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 which does not appear in the DDT entry is dropped, and the result is the final set of access rights which 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 subjects'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 subjects'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 which implements the restictions 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 01, 02, 03. It is also necessary to have a "gatekeeeper" domain P4 for use when changes in integrity level are required.

In order to implement the Strict Integrity Policy, we need only construct a DDT configuration as follows:

Object Type:	01	02	03
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)

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 polices must appeal to "trust" in order to accomodate practical processing requirements.

Assured Pipelines

We will now show that the assured pipleline structure can be readily accomodated by the type enforcement policy. We will show DDT and DTT configurations based on the following

Types: Unlabelled and Labelled data.

Domains: User, Labeller, and Output.

Unlabelled data is data which has only internal labels associated with it. Labelled data is data which 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 which 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 accesible to human analysis, understanding, and criticism.

This goal has led us to avoid the machine-generated proofs of previous efforts in favor of proofs which 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 hardcopy.

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 which 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 which 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.)

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 which has been shown to subsume both the practical variations of hierarchical integrity polices and cases which such polices 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.

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," these proceedings.

ACKNOWLEDGEMENTS

This effort has been supported by US Government Contracts MDA904-82-C-0444 and MDA904-84-C-6011.