A CAUTIONARY NOTE REGARDING THE DATA INTEGRITY CAPACITY OF CERTAIN SECURE SYSTEMS

Cynthia E. Irvine
Naval Postgraduate School
irvine@cs.nps.navy.mil

Timothy E. Levin
Naval Postgraduate School
levin@cs.nps.navy.mil

Abstract The need to provide standard commercial-grade productivity applications as the general purpose user interface to high-assurance data processing environments is compelling, and has resulted in proposals for several different types of "trusted" systems. We characterize some of these systems as a class of architecture. We discuss the general integrity property that *systems can only be trusted to manage modifiable data whose integrity is at or below that of their interface components.* One effect of this property is that in terms of integrity these *hybrid-security* systems are only applicable to processing environments where the integrity of data is consistent with that of low-assurance software. Several examples are provided of hybrid-security systems subject to these limitations.

Keywords: integrity, confidentiality, integrity capacity, secure system, multi-level security

1. Introduction

Data integrity is defined as "the property that data has not been exposed to accidental or malicious alteration or destruction." [29] A common interpretation is that high integrity information can be *relied* upon as the basis for critical decisions. However, the protection of high-integrity data in commercial systems has been both problematic to achieve and often misunderstood.

High Assurance Systems are designed to ensure the enforcement of policies to protect the confidentiality and integrity of information. To date, high-assurance systems have been expensive to produce and often lack support for, or compatibility with, standardized user-level applications. *Hybrid security* systems are intended to provide some desired functionality with high assurance of correct policy enforcement by utilizing a combination of high-assurance policy-enforcement components and low-assurance user interface and application components, thus addressing both the expense and compatibility problems typical of high-assurance systems.

In an era when users demand the productivity enhancements afforded by commercial software application suites, hybrid security architectures are of particular interest. Extensive study has demonstrated that hybrid security architectures using commercial user interface components can correctly enforce intended confidentiality policies, e.g. [25]. Less attention has been directed toward the effect of commercial user interface software on the integrity of data managed through those interfaces. Concerns include the integrity of the data modified using these commercial interfaces and then stored by high assurance components, as well as the integrity of data read from high assurance repositories and displayed to users.

While some developers have indicated that this problem is something "we have always known about," the problem may not be fully appreciated by the consumers of these systems. Our premise is that builders and buyers of systems designed to provide high assurance enforcement of security policies should be aware of the impact of component and architectural choices on the integrity of data that users intend to protect. Although the problem is exacerbated in systems designed to implement mandatory integrity models, such as represented by the Biba model [8], it is also significant in systems intended to support confidentiality policies. The former systems have explicit integrity requirements, whereas the latter may have implicit integrity expectations.

1.1 Contributions of this Paper

There is a large body of existing literature regarding integrity enforcement, requirements, and models; most of these address access control and related integrity issues, but do not address integrity capacity problems of system composition.

The National Research Council report on "Trust in Cyberspace" [31] identifies the construction of trustworthy systems from untrustworthy components as a "holy grail" for developers of trustworthy systems. And a National Computer Security Center guideline[30] that addresses both integrity and confidentiality issues, states that "the ability to run untrusted applications on top of TCBs [1] without undue loss of security is one of the major tenets of trusted computer systems." One of the primary results of our paper is to clarify the limitations of a significant class of these approaches with respect to integrity.

In this paper we examine integrity capabilities of component-based systems and provide a rigorous definition of *system integrity capacity.* This definition can form a basis for reasoning about systems with respect to their suitability for integrity policy enforcement. We also provide examples of several contemporary research-level security systems that exhibit the integrity capacity problem.

Finally, we provide a general conclusion regarding the integrity limitations of hybrid-security system composition: namely, system composition is problematic with respect to maintenance of high integrity data when utilizing commercial-grade products for user interfaces and applications.

1.2 Organization

The remainder of this paper is organized as follows. Section 2 provides a brief discussion of some related efforts involving security and integrity. We review concepts associated with confidentiality, integrity, and assurance in Section 3. Integrity considerations regarding system components and abstract subjects are discussed in Section 4. Section 5 presents the notion of "system integrity capacity," and Section 6 provides a derivation of this capacity for *hybrid security* systems, several of which are described. Our conclusions, in Section 7, complete the body of the paper. A discussion of malicious artifacts in commercial systems is included in Appendix 7.

2. Related Work

The architectural integrity issues we discuss have been addressed only indirectly in the literature. For example, the Seaview papers ([13], etc.) make it clear that the reference monitor will enforce integrity constraints on its subjects, such as the relational database management component; however, they do not explain that the use of a B1-level [2] RDBMS com-

[1]Trusted Computing Base[28]

[2]The terms used in this paper to reflect the evaluation class of systems and components are taken from [28] (e.g., B1 and B2) and [2] (e.g., EAL5).

ponent as the interface to users will limit the integrity range of data that the system can support.

The key issue addressed in our paper is how a system manages modifiable data. The Biba integrity model includes the restriction that a subject may modify an object only if the subject's integrity label "dominates" the object's integrity label. This and related characteristics of the "strict" integrity model are discussed extensively in the literature, starting with [8].

"Program integrity" [39] is related to strict Biba integrity, encompassing all of the restrictions of Biba integrity except for those related to the reading of files, while retaining restrictions for the execution of files. Strict integrity treats execution as a form of reading, whereas program integrity treats them separately [36]. Program integrity is of interest because it can be enforced with simple ring-bracket mechanisms [37], and results in "dominance" or "protection" domains, which can be used to enforce the relationships between the components, subjects and objects discussed in Section 4.

Lipner [24] applies Biba integrity to a real-world business scenario, working through the consistent application of hypothetical integrity labels in the context of a Biba mechanism to protect commercial data from unauthorized modification. This presentation does not address system level integrity problems resulting from the utilization of components with various integrity/assurance levels.

In contrast to low water-mark models, e.g. as discussed in [8], which address changes to the integrity level of a subject as it accesses objects with various integrity levels, we examine how the integrity value of data is affected as it is passed through data-modifying components with heterogeneous integrity properties.

Boebert and Kain [9] recognize the asymmetry of confidentiality and integrity, and remark on the vulnerability of information to corruption when only program integrity is enforced. Their work focussed on the use of domain and type enforcement to construct "assured pipelines" where the integrity level of data is changed as it moves through the pipeline. It does not discuss how software could present intrinsic limitations on the integrity of data to be processed.

Clark and Wilson [11] present a model for the protection of data integrity in commercial systems. In that model, components that modify data or handle user input must be "certified." However, their model does not address the relative integrity of the components and the data, nor does it address the resulting limits to the integrity of data that could be processed by such a system.

With respect to the problem of how to determine integrity labels for objects, Amoroso [4] relates evaluation assurance to software integrity, describing a broad range of (integrity) classes for articulating software trust. Karger suggests that a representation of literal evaluation levels could be used for integrity labels[19].

3. Background

This section sets the context for the presentation of system integrity capacity and attendant problems. Several concepts are examined in relation to integrity, including confidentiality, data versus code, assurance and trust, and multilevel security and the Biba model.

3.1 Integrity and Confidentiality

A given piece of information will have a *confidentiality value* as well as a separate *integrity value* . That is, there will be separately measurable effects (e.g., harm to the information owner) from the leakage vs. the corruption of the information. This is the case whether or not the data has been explicitly *labeled* with confidentiality and integrity designations (as is done in a multilevel-secure system). These labels may indicate both the degree with which we intend to protect the data as well as our assessment of the data's intrinsic value or sensitivity. The *labels* may or may not correspond to the actual integrity or confidentiality *value* of the data (in general, multilevel security models address the security values of the data; whereas the security labels are an implementation issue).

Integrity is, in many ways, the "dual" of confidentiality. Both integrity and confidentiality policies can be represented with labels that represent equivalence classes whose relationships form a lattice [41, 14]. Access control policy decisions can be based on the relative position of labels within a given lattice. Increasing the confidentiality "level" given to a subject (e.g., user) generally *expands* the set of objects that the subject may view; but an increase in integrity may *contract* the set of objects that a subject may view. Such semantic "inversions," and the sometimes non-symmetric nature of integrity and confidentiality properties (e.g., see [9]) can make their differences difficult to reason about. As a result, the analysis of integrity may be overlooked or avoided during the system design or acquisition process, in favor of more familiar confidentiality analyses.

3.2 Integrity of Data and Code

Integrity values are associated with *executable* software (e.g., programs, object code, code modules, components) as well as with *passive* data. In both cases, the integrity value relates to how well the data or program corresponds to its uncorrupted/unmodified original value (e.g., manufactured, installed, or shipped image). For programs, integrity also describes how well a program's behavior corresponds to its *intended behavior* (e.g., documented functionality or design documentation), including the notion that the code does not provide functionality beyond that which was intended (e.g., contain hidden behavioral artifacts). So, "integrity" means that the code has been unaltered, or is faithful to its origin, in both of these ways.

3.3 Assurance and Trust

Integrity of code is also closely related to "assurance" and "trust." Products that have been through security evaluations [28][2] receive an assurance-level designation. A methodical, high-assurance development process may produce code with fewer flaws, and consequently, behavior that is closer to that which is intended, than a low-assurance development process. Suitable security mechanisms and practices must also be in place to ensure the ability of the system to protect itself and provide continued system integrity during operation. This reliable code is sometimes called, or labeled, "high integrity;" it is also referred to as, "high assurance" code. Based on this designation, the product may be deemed suitable for handling data within a certain confidentiality or integrity range. Systems or components with demonstrated capabilities for security policy enforcement are sometimes called "trusted."

3.4 Multilevel Security

Multilevel systems partition data into equivalence classes that are identified by security labels. Data of different sensitivities is stored in different equivalence classes, such that (the data in) some equivalence classes are "more sensitive than," "more reliable than," or "dominate" (the data in) other equivalence classes. The dominance relation forms a lattice with respect to the labels/classes, assuming the existence of labels for universal greatest lower bound, GLB, and universal least upper bound, LUB. A reference validation mechanism (RVM, see "multilevel management component" in Figures 1, 2 and 3), mediates access to objects, controlling object creation, storage, access and I/O, thereby preventing policy-violating data "leakage" across equivalence classes. For confidentiality policy enforcement, a subject's (e.g., program or component's) ability to write-down or read-up is prevented with respect to the dominance relationship on *confidentiality* labels; for Biba-model integrity, read-down and write-up are prevented with respect to the dominance relationship on *integrity* labels. Most multilevel systems today are designed to enforce confidentiality constraints; some of these are also designed to constrain flow between integrity equivalence classes.

4. Integrity of Components and Subjects

The purpose of this section is to examine how integrity is interpreted with respect to the fundamental building blocks of secure systems.

The abstract architecture we are interested in is one of distributed storage, processing, and interconnection "components." A component is a functional system-level building block made up of software, firmware, hardware or any combination of these. Multiple components may reside on a single computer, but for simplicity's sake, we will assume that a single component does not encompass multiple remotely-coupled computers. Examples of components are shown in Section 6, and include a relational database management system, a security kernel, a client user application, an application server, and a graphical user interface program.

A component can include multiple code modules. The modules may be linked within a process, statically by a compiler/linker, or may have a more dynamic, runtime, linkage. A component can also encompass multiple processes, interconnected through message-passing, remote invocation, or other mechanisms.

Subjects are a modeling abstraction for reasoning about the security behavior of active computer elements such as programs and processes. A primary criteria for identifying a set of active computer elements together as a subject is that each subject has identifiable security attributes (e.g., identity and security level) that are distinct from other subjects. If the security attributes change over time, the elements are sometimes modeled as a different subject.

A component may manifest one or more subjects at a time. Each subject may encompass one or more of the component's modules, for example when they are linked within the same process. In a monolithic architecture, subjects may be identified with separate rings, or privilege levels, of a process [28], especially if the program has different security characteristics in the different rings. Typical systems support confidentiality and integrity labels for abstract subjects that are distinct from the labels on related components and modules (an alternative design would be to derive the subject label directly from the fixed component label). For example, the assignment of a subject label may be a mapping from the user's current "session level" to the subject representing the user. There are semantic limitations on this assignment with respect to the integrity level of the related modules and components.

4.1 Relation of Component and Subject Integrity

First we consider confidentiality. A subject may be associated with a particular confidentiality equivalence class for enforcement of mandatory access control. The mandatory confidentiality policy is not concerned with what happens between subjects and objects that are in the same confidentiality equivalence class: all reads and writes are allowed. The confidentiality policy is only concerned with what happens when a subject attempts to access an object in another equivalence class. We can interpret the subject reading data from a (source) equivalence class as moving data from that source into the subject's (destination) equivalence class, and writing to a (destination) equivalence class as moving data from the subject's (source) equivalence class to the destination equivalence class. The confidentiality policy says that when data is moved, the confidentiality label of the destination must always dominate the confidentiality of the source (again, data cannot move down in confidentiality).

In contrast, while the integrity policy, too, is concerned with movement of data across equivalence classes (the integrity label of the source must dominate the integrity of the destination), this policy is also concerned with the correctness of modifications, such that even if the subject is in the same equivalence class as the destination object, the modification must be that which has been requested: the *allowed* (e.g., intra-equivalence-class) modifications must be the *correct*, intended, modifications. The tacit assumption in integrity-enforcing systems is that the subject performs the correct modification (only) to its level of integrity (or assurance, if you will). Since an abstract subject's behavior is defined by its code, for coherent enforcement of integrity, the level of integrity assigned to the subject must be no higher than the integrity value of its code.

Components may not always receive an explicit security label, even in a system with labeled modules and other objects. Components may be composed of modules with different security labels. It is conceivable that a given component could be composed of both high-integrity and low-integrity modules, such that subjects with different integrity are supported by only modules of that same integrity. This would conform to the requirement stated above that a subject's integrity should be no greater than the integrity of its code. However, *most commercial components are not constructed this way. The simplifying assumption for this analysis is that modules within a given component are homogeneous with respect to their integrity, and the integrity of a component is the same as the integrity of its constituent modules.* Thus, we can can generalize the stated requirement to be that the level of integrity assigned to an abstract subject must be no greater than the integrity of the component that manifests the subject.

Combining this component-subject integrity relationship with the subject-object integrity relationship required for data modification (as per the Biba model, above), we arrive at a transitive relationship between the integrity of components and the objects which they access:

Given the sets of Components, Subjects, and Objects, where each subject in Subjects is an element of one component:

$$\begin{gathered}\forall\, c \in Components\ ,\ s \in Subjects\ ,\ o \in Objects :\\ current\_access(s, o, modify)\ and\ s \in c\ \Rightarrow\\ integrity(c)\ \geq\ integrity(s)\ \geq\ integrity(o)\end{gathered}$$

Systems that enforce integrity policies are generally intended to automatically ensure the correct relationship between the integrity level of subjects and the integrity level of accessed objects. However, the enforcement of the relationship between a subject's integrity and its component's integrity may be less clear. Some systems may be able to enforce the relationship between the integrity of subjects and their related modules. For example, this could be enforced by Biba-like labels on executables or other program integrity mechanisms such as rings [38] and ring brackets [1] which can also be represented as attributes on system elements. If these relationships are not enforced during runtime, then the correct relationships may need to be maintained by social convention/procedure.

The relationship between the integrity of a component's subjects and the integrity of the non-software portion of the component is also enforced via social convention (again, component integrity must dominate the subject integrity).

4.2 Component Integrity Labels

This leaves the question of correct integrity labeling of components (and modules). Confidentiality and integrity labels of passive data objects can be correctly assigned based on the data owner's perception of the object's sensitivity (e.g., harm caused by unauthorized disclosure or modification).

For active objects (viz, code rather than data) integrity labels, as well as confidentiality labels, are usually assigned by the system or network security designer to maximize system security and functionality while being consistent with the principle of least privilege [35]. Best judgment may play a large role in this assignment. For example, if a monolithically-compiled software component is made of up diversely-assured internal modules, it may be the responsibility of a designer, integrator or configuration manager, as stipulated by social convention or procedures, to assign an appropriate integrity level to the executable component. However, the pedigree of the code establishes a real-world limit to the integrity label that can be associated with a component. Intuitively, code that has unknown integrity characteristics, e.g., it is found on the street, should not be accorded a high-integrity label.

The "Yellow Book"[27] is an example of a scheme for determining confidentiality ranges based on the evaluation or assurance level of the components involved, where higher assurance components are allowed to be associated with greater confidentiality ranges. However, there is no "Yellow Book" for integrity to show what integrity label should be allowed or inferred for a code component based on its evaluation/assurance level, although some schemes have been suggested[4, 19].

4.3 Commercial Application Component Integrity

Commercial application components are of particular interest with respect to correct integrity labeling in hybrid security architecture systems (see Section 6). We define *commercial application components* to have been either unevaluated with respect to security policy enforcement, or evaluated below Class B2/EAL5[3]. In the security and evaluation community, components evaluated below B2/EAL5 have historically been considered to be "low assurance" (see, for example, [22]). This is so for several reasons [28, 2]:

- Weak developmental assurance, for example to ensure that unintended malicious artifacts (e.g., Trojan horses and trap doors) are not inserted during manufacture. There is no or very little requirement for system configuration management. There is no requirement for configuration management of development tools.

[3]As there have been few, if any, commercial applications evaluated at B2 or higher, we consider this to be a conservative, non-exclusionary, definition.

- Little or no code analysis, and *no* examination of code for malicious artifacts after manufacture (i.e., during evaluation). There is no requirement for code correspondence [4] to the system specification or for justification of non-policy-enforcing modules. There is no requirement for internal structure (e.g., modularity or minimization) which would enable the meaningful analysis of code functionality.
- Weak assurance that malicious artifacts are not inserted after manufacture. For example, there is no requirement for trusted distribution procedures: no assurance that the system delivered to the end customer is in fact the intended or specified system.

Recall that the semantics of a code integrity label includes an indication of how its behavior corresponds to an intended (e.g., specified) behavior. The fact that there is little assurance that code that has been evaluated below B2/EAL5 functions (only) the way it is supposed to, indicates that there must be a corresponding limit to the value of an integrity label associated with such code (see Appendix 7). We will call this integrity limit, nominally, "low assurance," and assert that components evaluated below B2/EAL5 should be labeled at this, or some lower level. Similarly, code that has not been evaluated at all would be attributed with a (nominal) "no assurance" integrity label. The names of these two labels or the precise evaluation class names are not significant; rather, it is significant to the maintenance of data integrity in hybrid security systems that site security managers/administrators, data owners, and other security policy stake-holders understand the integrity value of their systems' components and of the data entrusted to these systems.

5. Security System Data Capacities

In this section, the notion of *system integrity capacity* will be introduced. This term relates to the ability of a system to handle high-integrity data.

The network architecture of a multilevel system can help to ensure that the actions of other components are constrained by its RVM, for example, through limiting the interconnections or data paths allowed between components. In the architectures discussed in this paper, the separation of data is maintained by either: (1) partitioning the data (and processing elements) into distinct physical equivalence classes and using the RVM to ensure that the security level of the user session matches

[4]Mapping of each specified function to the code that implements it, and accounting for unmapped code.

the security level of the class with which it is connected (e.g., Figure 2), or (2) using the RVM to logically partition the different data equivalence classes and to match the user session level to only the appropriate domain(s) (see Figures 1 and 3).

Our central question is, "for what range of *user data* [5] can we trust such a multilevel system, or any system, to maintain data separation?" Clearly, we would not want to trust a very weak system to protect/separate very highly sensitive information. While our focus is on integrity-related issues, for comparison we will examine cases of both confidentiality and integrity.

5.1 Confidentiality Capacity

For confidentiality, *a multilevel system can be trusted to manage data to the confidentiality range of its RVM.* We call this the *system confidentiality capacity.* For example, if the system's RVM component is assigned or is otherwise deemed capable of managing a range from Unclassified to Secret, we can say the system as a whole is trusted to handle data in that range. This is because the RVM will constrain the actions of the other components to not leak data across equivalence classes, regardless of the level of trust we have in those other components (given a coherent network architecture). To state confidentiality capacity more formally, consider a system, C, comprising a set of components, $\{c\}$, and let RVM be a component in C that enforces the confidentiality policy on other components. Then,

$$c\_capacity(C) = c\_capacity(RVM)$$

5.2 Integrity Capacity

For integrity, on the other hand, *a system can be trusted to manage modifiable data (only) to the integrity limit of its interface components*, where interface components include the various graphical user interfaces and data management applications through which users' data must pass. This is the "system integrity capacity."

System integrity capacity is different from (i.e., not the "dual" of) system confidentiality capacity because we assume that a component will handle modification of objects correctly, only to its level of integrity/assurance. For confidentiality, even if a non-RVM component were infected with malicious code, it could not exfiltrate the information across the equivalence-class boundary, because the RVM component won't let that happen. However, for integrity, once the component has

[5] The ability of a system to protect and maintain *system* data is not addressed in this paper.

approval for modify access, the RVM is powerless to ensure that the correct, and only the correct, modifications are made. Therefore, the assurance level of the individual (viz, non-RVM) component has bearing on its assigned integrity label, but is not necessarily relevant to its assigned confidentiality label.

The input and output mechanisms of a computer system limit the quantity and quality of information that flows through the system, just as the in- and out-flow of water and electricity are limited in hydraulic and electrical systems by their interface devices.

For computers, the I/O mechanisms and related applications, by definition, handle all data entering and leaving the system. Where those mechanisms and applications are configured to be able to modify data, they can potentially effect the integrity of the data entering and leaving the system. The nature of this effect is as follows.

Definitions

C	:	the universal set of components $\{c_1, c_2, \cdots, c_n\}$
O	:	the universal set of objects $\{o_1, o_2, \cdots, o_m\}$
INTEGRITY	:	a lattice of integrity levels: $\{integrity_1, integrity_2, \cdots, integrity_q\}$
modify	:	a relation that defines the fact that a component $c \in C$ has been used to to modify an object $o \in O$
$\mathcal{SYS}$	:	a system comprised of a set of components $c \in C$

Axiom 1

A modified datum is either unchanged in integrity, or takes on an inherent integrity value dominated by the integrity of the data-modifying component.

$$\forall\, c \in \mathcal{SYS},\ o \in O:\ \ modify(c, o) \Rightarrow integrity(c) \geq integrity(o')$$

For example, if a "certified" datum is modified by an "untrusted" code component, the modified datum becomes at best "untrusted," assuming that "certified" dominates "untrusted." If an "untrusted" datum is modified by a "certified" component, the datum becomes at best "certified," indicating it might have been upgraded in integrity.

For a high-assurance integrity-enforcing system, subjects, including the applications that manage user I/O, will be limited by the RVM from modifying protected objects that are above the subject's integrity level. However, if the application is responsible for passing data from one of those objects to, for example, an output device like the computer screen, then the application can simply modify the data in passing without modifying the source object.

Similarly, even if a component does not modify the data directly, it may request that the modification be done by another component, for example, where a user interface component requests from another (e.g., remote) component that an object be created on behalf of the user. Since the requesting component might request the *wrong* modification, we consider it to be a "data-modifying" component. So a system's "data-modifying" components are those components that are able to modify or control the modification of user data. In general, all interface components and other components on the "path" between the user who requests a data access and the ultimate data source (for data reads) or destination (for data writes) are "data-modifying" components, unless they can be guaranteed to not modify, create or delete user data objects or control such operations[6].

Therefore, even for systems that enforce integrity policies, a computer system can only be trusted to manage modifiable data whose integrity is at or below that of its user interface and application components. This is true even if the data is either (1) integrity-upgraded internally by various components, (2) "hand installed" into high integrity internal objects, or (3) imported from specialized high integrity sensing devices, since to be useful, the data will once again be "handled" by the standard interface and application components for access by users. We will note that, theoretically, manual procedures, such as visual inspection of data items retrieved from a hybrid security system, could be used to ensure that processing corruption has not occured, however, this is not generally feasible in commercial or production environments.

As a group, then, the interface components and associated applications determine the integrity limit of the data that a system can handle (*i_capacity*). The interface components are a subset of $\mathcal{SYS}$, indicated $\mathcal{SYS}_{interface}$, and the highest integrity data obtainable from a system $\mathcal{SYS}$ is by way of the user interface component with the highest integrity (viz, the least upper bound of the integrity of all interface components).

$$i\_capacity(\mathcal{SYS}) \ = \underset{c \,\in\, \mathcal{SYS}_{interface}}{\mathrm{LUB}} \left(integrity(c) \right)$$

This gives a "best case" analysis for the integrity that we might expect a system to handle. For example, a high integrity interface application, were it to be available, dependent upon a low integrity database, would not normally improve the integrity of data returned from the database to

[6]The "control" part of this definition makes it broader than the Bell and LaPadula[7] concept of "current access," which indicates only objects with direct access to data.

the user, although this expression of system integrity capacity would indicate that the data accessed through the high integrity interface might be of high integrity. The general case is that the *i_capacity* expression must allow for such upgrades. However, not all systems are designed for data integrity upgrades. A more conservative axiom regarding modification, which does not consider upgrading, results in an *i_capacity* based on the lowest integrity of the components in each path.

Axiom 2

A modified datum takes on an inherent integrity value that is the greatest lower bound of the data and the data-modifying component.

$$\begin{gathered}\forall\, c \in C\ ,\ o \in O : modify(c, o) \Rightarrow \\ integrity(o') = GLB(integrity(c), integrity(o))\end{gathered}$$

We now define an individual *data transfer* within the system, a *path* through the system, and the integrity of such a path.

trans: A relation on $C \times C$ that defines an individual transfer of data between components. Data is passed directly from the origin component, c_i, to the terminus component, c_j:

$$c_i\_trans\_c_j$$

path: A sequence of $trans$ relations such that for every pair of consecutive relations ($c_i\_trans\_c_j$, $c_j\_trans\_c_k$), the *terminus* of the first and the *origin* of the second coincide[34]. For example, this is a path with n relations: $< c_0\_trans_1\_c_1,\ c_1\_trans_2\_c_2, \cdots,\ c_{n-1}\_trans_n\_c_n >$

The integrity of a path is the greatest lower bound of the components in the path:

$$integrity(path) \ = \ \underset{c\,\in\,path}{\mathrm{GLB}} \left(integrity(c) \right)$$

Given these definitions, we provide the alternative, more conservative, expression for *i_capacity*.

Let $\pi(\mathcal{SYS})$ be the set of all paths in SYS whose origin or terminus is in $\mathcal{SYS}_{interface}$, then:

$$i\_capacity(\mathcal{SYS}) \ = \ \underset{path\,\in\,\pi(\mathcal{SYS})}{\mathrm{LUB}} \left(integrity(path) \right)$$

6. Hybrid Security Systems

The systems we are concerned with are those that combine low-assurance commercial components and specialized (e.g., high-assurance) multilevel components specifically to enforce mandatory security policies while using commercial user-level interfaces and applications. These systems, as a class, are composed of the following components:

- commercial terminals or workstations
- commercial user interfaces, applications and application servers
- Storage devices containing multiple levels of data
- Multilevel-management components
- TCB Extensions
- commercial network interconnections

The interested reader is referred to [17] for a detailed description of these components. Of particular note, however, is the description of applications. In the generic "hybrid security" architecture defined in [17], applications interface with the user and participate in the management of all user data. Specifically, the application components have the ability to modify data on behalf of the user (which is to say that read-only systems are not of interest). The general functionality of commercial applications such as word processing, spread sheet, slide presentation, time management, and database tools indicate that, to be useful, they are intended to modify, as well as read, data.

To illustrate the relevance of our concerns for the handling of high integrity data in hybrid security systems, we describe here several systems from the security literature that exhibit dependence on the integrity of commercial components.

A non-distributed version of the model architecture is shown in Figure 1. In this layout, the component interconnections consist of process-internal communications. The lowest layer (viz, "ring") of the process is a multilevel kernel or operating system, with an application (e.g., multilevel-aware RDBMS) and user interface in higher layers. A separate process is created for each security level. An example of this version of the architecture is that of the Seaview project [15, 25], and "Purple Penelope" [33] (the latter includes a degenerate case of a RVM). A variation on this theme is the trusted Virtual Machine Monitor (VMM) architecture, in which a separate version of the OS, in addition to the application and user interface, is created at each security level[20, 26, 6], and multilevel management occurs below that in the VMM layer.

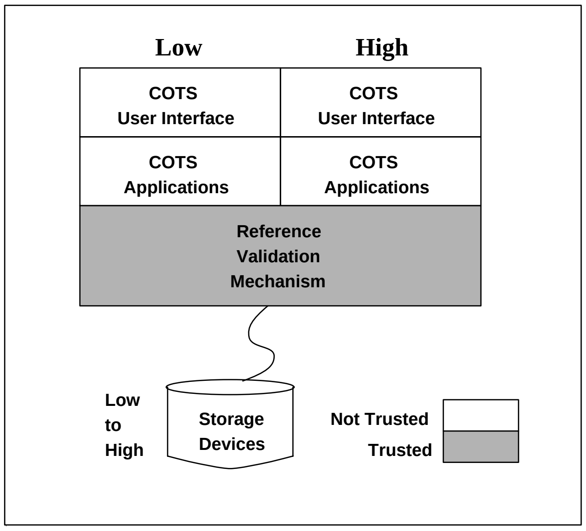

Figure 1. Single Process Architecture (Network Connections are degenerate.)

A simple distributed instantiation is shown in Figure 2. Here, there are logically separate single-level workstations connected by a switch to data management subsystems at different (single) levels. Software associated with the switch ensures that the current level of the workstation matches the level of data subsystem indicated by the switch setting. An example of this version of the architecture is that of the Starlight project [5] (Starlight may allow low confidentiality information to flow through the switch to high sessions, providing "read-down" capability).

The third instantiation of the model architecture is shown in Figure 3. In this layout, there are logically separate single-level terminals (multiplexed onto one physical terminal by purging of state between session-level changes) connected via TCB extensions to multilevel-aware application server(s) running on the multilevel (TCB) component. An example of this version of the architecture is that of the Naval Postgraduate School's Monterey Secure Architecture (MYSEA) system, based on [18].

6.1 Integrity Capacity of Hybrid Security Systems

Based on the preceding discussion, the system integrity capacity of *hybrid security* systems can be summarized as follows:

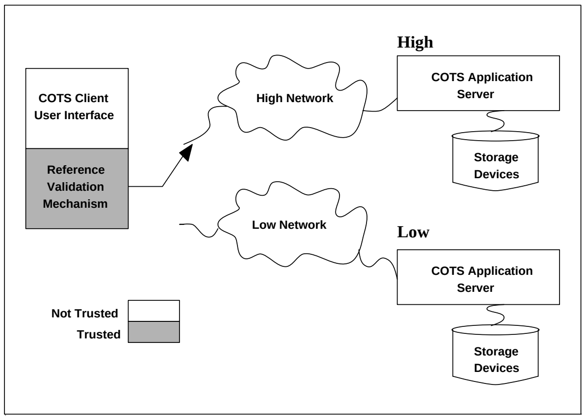

Figure 2. Switch-Based Architecture

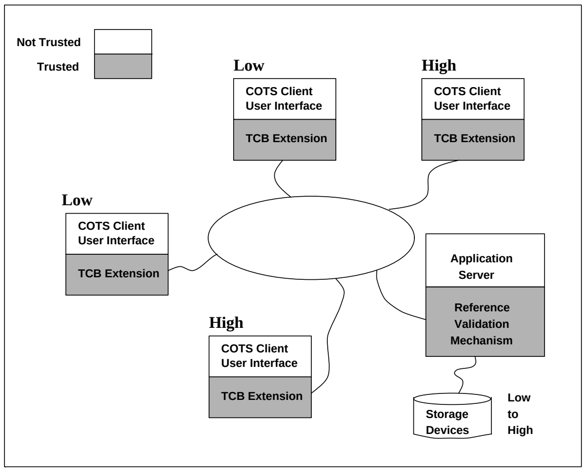
Not Trusted
Trusted
Low
COTS Client User Interface
TCB Extension
High
COTS Client User Interface
TCB Extension
Low
COTS Client User Interface
TCB Extension
Application Server
Reference Validation Mechanism
High
COTS Client User Interface
TCB Extension
Storage Devices
Low to High

Figure 3. Distributed Multilevel Server Architecture

- A system's integrity capacity is the LUB of the integrity of its interface/application components.
- All interface/application components in hybrid security systems are commercial
- Commercial interface/application components are of "no assurance" or "low assurance" integrity.

- Therefore, the system integrity capacity of a *hybrid security* system is generally no higher than "no assurance" or "low assurance."

An implication of this conclusion is that *hybrid security architecture* systems are not suitable for automated information processing environments in which there are expectations or requirements to maintain data integrity above the nominal "no assurance" or "low assurance" level. Another implication is that composition of trusted systems utilizing only commercial products as interface components is problematic with respect to integrity.

7. Conclusion and Discussion

We have shown that the integrity of a computer system's interface components limits the data "integrity capacity" of the system. This is in contrast to the "confidentiality capacity" of a system, which is determined by characteristics of the system's policy-enforcement component(s), but is not dependent on the interface components.

We have discussed why commercial components should not be attributed with integrity properties above a certain "low-assurance" level, and that *hybrid security* systems should not be trusted with data whose integrity is above that level. An implication from this conclusion is that *hybrid security* systems are not suitable in computing environments where there is an expectation of maintaining data integrity above this basic, low-assurance level.

Situations where corrupted data could have significant consequences are:

- A legal setting where the "truth" of data might be questioned
- Handling of high integrity intelligence data for critical decision making
- The production of high assurance system components
- Systems where human life might be affected by improper execution of code
- High reliability embedded systems

We have concentrated on issues of integrity in multilevel secure systems, however, the distinctions we have made are germane to other systems where weak integrity components are utilized and stronger data integrity is expected.

One might say, "what difference does it make if a component has too high of an integrity label, and its real integrity value is low? These commercial software vendors can be generally trusted, since it is in their best interest to ship a product that does not corrupt data." This attitude reflects a common misunderstanding of data integrity enforcement. Certainly, most security analysts and engineers would agree that high-assurance policy-enforcement components are needed to safeguard the *confidentiality* of highly sensitive multilevel data; then, why would there be any lesser concern for the ability of a system to protect the *integrity* of highly sensitive data? From a more technical viewpoint, if a system's objects do not have data sensitivity (confidentiality and integrity) labels that match the objects' real sensitivity values, then the system does not correspond to its model, and its behavior may be undefined. Also, refer to Appendix 7, for a review of common "Subversive Artifacts in Commercial Software."

The result presented in this paper places a limit on what is achievable in system integrity architectures. Such a finding can help to refine the direction for constructive efforts and does not preclude the construction of useful systems any more than other negative results, e.g. [16, 21], have in the past.

One might also ask if high integrity is ever achievable. The answer is yes, but not with the type of commodity application components available today (viz, where commodity implies weak integrity of software functionality). Systems that *could* provide high integrity today are (1) a system composed entirely of high-assurance components, or (2) a system that protects high integrity data from modification by all but high-assurance components. Examples of the first are systems intended to perform safety-critical functions such as avionics and certain medical systems[23]. An example of the second is a client-server system composed of high-assurance client (e.g., web browser) and server components that encrypt their communication such that it is protected from modification during transit through low assurance network components (e.g., via a Virtual-Private-Network-style connection). As noted previously, such systems carry the expense of custom high-assurance development.

Appendix: Subversive Artifacts in Commercial Software

There is clear evidence that subversion of commercial software through hidden entry points (trap doors) and disguised functions (Trojan horses) is more common than generally perceived. Entire web sites [3] are devoted to describing clandestine code which may be activated using undocumented keystrokes in standard commercial applications. Sometimes this code merely displays a list of the software developers' names. Other times the effects are extremely elaborate as in the case of a flight simulator embedded in versions of the Microsoft Excel Spreadsheet software. That these "Easter Eggs" are merely the benign legacy of the programming team is perhaps a reflection of the general good intentions of the programmers. Malicious insertions, such as long-term time bombs, are just as easily possible.

An indication of the serious nature of the problem was provided in April 2000 when news reports created a mild hysteria surrounding the possibility of a trapdoor in the code of a widely used web server[32]. Subsequent investigations revealed that instead of a trapdoor, the code contained nasty remarks about corporate competitors and well as violations of company coding standards[12]. The fact remains, however, that when rumors of the trapdoor were initially published, few believed that artifices of this type were possible in such a popular software product. However, millions of users do not eliminate the problem of low integrity. Another example of the vulnerability of commercial source code occured in October 2000, when it was revealed that outsiders had access to the development environment of a major software vendor for some period of time[10].

In his Turing Prize Lecture, Ken Thompson described a trapdoor in an early version of the Unix operating system [40]. The cleverness of the artifice was evident in that the artifice was said to have been inserted into the operating system executable code by the compiler, which had been modified so that recompilations of the compiler itself would insert the trapdoor implantation mechanism into its executable while leaving no evidence of the trapdoor in the source code for either the operating system or the the compiler. The presence of this sort of trap door is speculative in *any* compiler and must be addressed through life-cycle assurance of tools chosen for high assurance system development.

References

[1] Gemini Trusted Network Processor (GTNP). In *Information Systems Security Products and Service Catalog Supplement*, Report No.CSC-PB-92/001. April 1992. 4-SUP-3a.3.

[2] ISO/IEC 15408 - Common Criteria for Information Technology Security Evaluation. Technical Report CCIB-98-026, May 1998.

[3] *The Easter Egg Archive.* http://www.eeggs.com/, last modified 19 May 2000.

[4] E. Amoroso, J. Watson, T. Nguyen, P. Lapiska, J. Weiss, and T. Star. Toward an approach to measuring software trust. In *Proceedings 1991 IEEE Symposium on Security and Privacy*, pages 198–218, Oakland, CA, 1991. IEEE Computer Society Press.

[5] M. Anderson, C. North, J. Griffin, R. Milner, J. Yesberg, and K. Yiu. Starlight: Interactive Link. In *Proceedings 12th Computer Security Applications Conference*, San Diego, CA, December 1996.

[6] S. Balmer and C. Irvine. Analysis of Terminal Server Architectures for Thin Clinents in a High Assurance Network. In *Proceedings of the 23rd National Information Systems Security Conference*, pages 192–202, Baltimore, MD, October 2000.

[7] D. E. Bell and L. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. Technical Report M74-244, MITRE Corp., Bedford, MA, 1973.

[8] K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report ESD-TR-76-372, MITRE Corp., 1977.

[9] W. Boebert and R. Kain. A practical alternative to hierarchical integrity policies. In *Proceedings 8th DoD/NBS Computer Security Conference*, pages 18–27, Gaithersburg, MD, September 1985.

[10] T. Bridis, R. Bickman, and G. Fields. Microsoft Said Hackers Failed to See Codes for Its Most Popular Products. http://interactive.wsj.com/archive/retrieve.cgi?id=SB972663334793858544.djm, October 2000.

[11] D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings 1987 IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, CA, April 1987. IEEE Computer Society Press.

[12] R. Cooper. Re: Security experts discover rogue code in Microsoft software. http://catless.ncl.ac.uk/Risks/20.88.html#subj11, May 2000.

[13] D. Denning, T. F. Lunt, R. R. Schell, W. Shockley, and M. Heckman. The seaview security model. In *Proceedings 1988 IEEE Symposium on Security and Privacy*, pages 218–233, Oakland, CA, April 1988. IEEE Computer Society Press.

[14] D. E. Denning. *Secure Information Flow in Computer Systems.* PhD thesis, Purdue Univeristy, West Lafayette, IN, May 1975.

[15] D. E. Denning, T. F. Lunt, R. R. Schell, W. Shockley, and M. Heckman. Security policy and interpretation for a class a1 multilevel secure relational database system. In *Proceedings 1988 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1988. IEEE Computer Society Press.

[16] M. Harrison, W. Ruzzo, and J. Ullman. Protection in Operating Systems. *Communications of the A.C.M.*, 19(8):461–471, 1976.

[17] C. Irvine and T. Levin. Data integrity limitations in highly secure systems. In *Proceedings of the International Systems Security Engineering Conference*, Orlando, FL, March 2001.

[18] C. E. Irvine, J. P. Anderson, D. Robb, and J. Hackerson. High Assurance Multilevel Services for Off-The-Shelf Workstation Applications. In *Proceedings of the 20th National Information Systems Security Conference*, pages 421–431, Crystal City, VA, October 1998.

[19] P. Karger, V. Austel, and D. Toll. A new mandatory security policy combining secrecy and integrity. Technical Report RC 21717(97406), IBM Research Division, Yorktown Heights, NY, March 2000.

[20] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn. A VMM Security Kernel for the VAX Architecture. In *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 2–19. IEEE Computer Society Press, 1990.

[21] B. Lampson. A Note on the Confinement Problem. *Communications of the A.C.M.*, 16(10):613–615, 1973.

[22] T. M. P. Lee. A Note on Compartmented Mode: To B2 or not B2? In *Proceedings of the 15th National Computer Security Conference*, pages 448–458, Baltimore, MD, October 1992.

[23] N. G. Levenson. *Safeware- System safety and Computers.* Addison-Wesley, 1995.

[24] S. B. Lipner. Non-Discretionary Controls for Commercial Applications . In *Proceedings 1982 IEEE Symposium on Security and Privacy*, pages 2–20, Oakland, 1982. IEEE Computer Society Press.

[25] T. F. Lunt, R. R. Schell, W. Shockley, M. Heckman, and D. Warren. A Near-Term Design for the SeaView Multilevel Database System. In *Proceedings 1988 IEEE Symposium on Security and Privacy*, pages 234–244, Oakland, 1988. IEEE Computer Society Press.

[26] R. Meushaw and D. Simard. Nettop. *Tech Trend Notes*, 9(4):3–10, Fall 2000.

[27] National Computer Security Center. *Computer Security Requirements, Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments*, CSC-STD-003-85, June 1985.

[28] National Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, December 1985.

[29] National Computer Security Center. *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, NCSC-TG-005, July 1987.

[30] National Computer Security Center. *A Guide to Understanding Covert Channel Analysis of Trusted Systems*, NCSC-TG-030, November 1993.

[31] National Research Council. *Trust in Cyberspace*, Washington, DC, 1999. National Academy Press.

[32] Newsscan.com. Security experts discover rogue code in Microsoft software. http://catless.ncl.ac.uk/Risks/20.87.html#subj8, April 2000.

[33] B. Pomeroy and S. Weisman. Private Desktops and Shared Store. In *Proceedings 14th Computer Security Applications Conference*, pages 190–200, Phoenix, AZ, December 1998.

[34] F. P. Preparata and R. T. Yeh. *Introduction to Discrete Structures*. Addison-Wesley Publishing, Co., Reading, MA, 1973.

[35] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[36] R. Schell and D. Denning. Integrity in trusted database systems. In *Proceedings 9th DoD/NBS Computer Security Conference*, Gaithersburg, MD, September 1986.

[37] M. D. Schroeder, D. D. Clark, and J. H. Saltzer. The Multics Kernel Design Project. *Proceedings of Sixth A.C.M. Symposium on Operating System Principles*, pages 43–56, November 1977.

[38] M. D. Schroeder and J. H. Saltzer. A Hardware Architecture for Implementing Protection Rings. *Comm. A.C.M.*, 15(3):157–170, 1972.

[39] L. J. Shirley and R. R. Schell. Mechanism Sufficiency Validation by Assignment. In *Proceedings 1981 IEEE Symposium on Security and Privacy*, pages 26–32, Oakland, 1981. IEEE Computer Society Press.

[40] K. Thompson. Reflections on Trusting Trust . *Communications of the A.C.M.*, 27(8):761–763, 1984.

[41] K. B. Walter, W. F. Ogden, W. C. Rounds, F. T. Bradshaw, S. R. Ames, and D. G. Shumway. Primitive Models for Computer Security. In *Case Western Reserve University Report*, ESD-TR-74-117, January 1974. Electronic Systems Division, Air Force Systems Command.