NPS-CS-00-008

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Data Integrity Limitations in
in Hybrid Security Architectures

by

Cynthia E. Irvine
Timothy E. Levin

December 2000

Prepared for: Community Intelligence Office and
Defense Advanced Research Project Agency   Information Technology Organization

This page left intentionally blank

This report was prepared by the Naval Postgraduate School Center for Information Systems Security (INFOSEC) Studies and Research (NPS CISR) . Support for this work was provided by the Defense Advanced Research Projects Agency/Information Technology Organization and the Community Intelligence Office.

This report was prepared by:

Cynthia E. Irvine
Assistant Professor

Timothy E. Levin
Senior Research Associate

Reviewed by:

Neil C. Rowe
Professor
Department of Computer Science

Released by:

D. C. Boger, Chair
Department of Computer Science

D. W. Netzer
Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE	Form approved OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** 7 December 2000	**3. REPORT TYPE AND DATES COVERED** Progress; 6/1/00 – 12/1/00

4. TITLE AND SUBTITLE Data Integrity Limitations in Hybrid Security Architectures	**5. FUNDING** MIPR No. 00-E583
6. AUTHOR(S) Cynthia E. Irvine and Timothy E. Levin	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Center for Information Systems Security Studies and Research (NPS CISR) Naval Postgraduate School, 833 Dyer Road, Monterey, CA 93943	**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-CS-00-008
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA/ITO 3701 North Fairfax Drive Arlington, VA 22203-1714 Community Intelligence Office Washington, D.C. 20505	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (Maximum 200 words.)

We discuss a class of computer/network architectures that supports multilevel security and commercial applications, while utilizing primarily commercial-off-the-shelf (COTS) workstations, operating systems and hardware components. We show that a property of these architectures is that, while they are capable of supporting multilevel confidentiality policies, they do not generally support partially ordered integrity policies: specifically, these architectures do not support the maintenance of data that is higher in integrity than the integrity level of the COTS components, thus limiting their applicability to processing environments where the integrity of data is consistent with that of low-assurance software.

14. SUBJECT TERMS Integrity, Confidentiality, Multilevel Security, Balanced Assurance, Integrity Capacity	**15. NUMBER OF PAGES** 20
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** Unclassified

The Data Integrity Capacity of Secure Systems

Cynthia E. Irvine
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943

Timothy E. Levin
Anteon Corporation
c/o Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943

Abstract

We discuss a class of computer/network architectures that supports multilevel security and commercial applications, while utilizing primarily commercial-off-the-shelf (COTS) workstations, operating systems and hardware components. We show that a property of these architectures is that, while they are capable of supporting multilevel confidentiality policies, they do not generally support partially ordered integrity policies: specifically, these architectures do not support the maintenance of data that is higher in integrity than the integrity level of the COTS components, thus limiting their applicability to processing environments where the integrity of data is consistent with that of low-assurance software.

1 Introduction

Data integrity is defined as "the property that data has not been exposed to accidental or malicious alteration or destruction."[28] 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.

Integrity is, in many ways, the "dual" of confidentiality. Both integrity and confidentiality policies can be represented with labels whose relationships form a lattice [41]. Access control policy decisions can be based on the relative position of labels within a given lattice. Whereas the level of "confidentiality" protection to be applied to a datum can be taken as a measure of how much harm its unauthorized *disclosure* would cause the owner, the required level of "integrity" can be viewed as how much harm would be caused by the unauthorized *modification* of the datum. 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 will $contract$ the set of objects that a subject may view. Such semantic "inversions," and the sometimes non-symmetric nature of integrity and confidentiality properties can make their differences difficult to reason about ([17] pages 68-71). 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.

Integrity of code is also closely related to “assurance” and “trust.” Products that have been through security evaluations [27][3] 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. More 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 range of confidentiality or integrity. Systems or components that have demonstrated capabilities for security policy enforcement are called “trusted.”

High assurance systems are designed to ensure the enforcement of policies to protect the confidentiality and integrity of information. The *hybrid security architecture systems* discussed in this paper are intended to provide some desired functionality with high assurance of correct policy enforcement by utilizing a combination of high-assurance and low- assurance components. The purpose of this paper is to shed some light on a subtle aspect of high-assurance system engineering that has not often been openly addressed by the developers of these systems: that is, a system can only be trusted to manage data whose integrity is at or below that of its weakest component. 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 non-discretionary security policies may ignore 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 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.

The remainder of this paper is organized as follows. Section 2 provides a brief discussion of some related efforts involving security and integrity. This is meant to provide a sampling, rather than a comprehensive review of preceding work, as the area is quite broad. We review concepts associated with confidentiality, integrity, and high-assurance systems in Section 3. A description of the model hybrid security architecture which we believe characterizes a variety of modern, distributed, secure system designs follows in Section 4. Section 5 presents the notion of “system integrity capacity,” and provides a derivation of this capacity for *hybrid security architecture* systems. Our conclusions, in Section 6, complete the body of the paper. An overview of requirements for achieving integrity in AIS components is presented in Appendix A; and a discussion of malicious artifacts in commercial systems is presented in B.

2 Related Work

Architecture-based integrity problems have been addressed only indirectly in the literature. For example, the Seaview papers ([12], 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 inclusion of a B1-level RDBMS component in the system will limit the integrity range of data that the system can support.

A key issue to be addressed here is how a system manages writable 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" [38] is related to strict Biba integrity discussed here, 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 [35]. Program integrity is of interest because it can be enforced with simple ring-bracket mechanisms [37], but since program integrity "write" restrictions are the same as for Biba integrity, it is not discussed further in this paper.

Amoroso [5] relates evaluation assurance to software integrity, describing a broad range of (integrity) classes for articulating software trust.

Aspects of this problem were presented to the Workshop on Innovations in Strong Access Control [18], and were recorded as a technical report [21].

3 Background

This section sets the context for the presentation of the model architecture and its attendant problems. Integrity is examined in relation to confidentiality, as well as in relation to both data and code. The basic Biba integrity model is described, and the motivations for development of hybrid security architectures are examined.

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 values (as is done in a multilevel-secure system).

Confidentiality and integrity policies may not be clearly differentiated in complex computer systems. In the military sector, secret information is often treated as reliable. Intelligence information of both high and low integrity is often handled as highly classified. Sometimes, classification is related to sources and methods. If the information is disconnected from its source, confidentiality may not be required, but a means of attesting for the reliability of the information may be needed.

In the simplest case, the confidentiality and integrity values are binary, high/low secrecy and high/low integrity, where high and low are in relation to the usual notion of a $dominance$ relationship for comparing relative levels of integrity. The values may be applied to information in different combinations, as show in the following examples:

- High Confidentiality and Low Integrity: for example, potentially critical information from an unreliable source. This type of (e.g., intelligence community) information may be held in strict confidentiality pending confirmation by a more reliable source.
- Low Confidentiality and High Integrity: for example, selected well established facts. The Oxford English Dictionary is considered to be a highly reliable (and definitive) source for definitions of words used by English speaking peoples.
- Low Confidentiality and Low Integrity: for example, the boasts of the children on the playground, or stock tips on an internet chat line.
- High Confidentiality and High Integrity: for example, trade-secret formulas, such as those used by chemical companies, recipes for a fast food vendor's "secret sauce" or the closed source code of a software vendor that provide specific functionality.

3.2 Data and Code

Integrity values may be associated with executable software (e.g., programs, code, code modules, components) as well as with passive data. In both cases the integrity level relates to how well the data or program corresponds to an *ideal* (e.g., original value or "as-shipped" image). For programs, integrity also describes how well a program's behavior corresponds to some ideal *behavior* (e.g., documented functionality), including the notion that the code does not provide functionality beyond that ideal (viz, contain hidden behavioral artifacts). So in our example above, of high confidentiality, high integrity closed source code of a software vendor, "integrity" refers to the level of its protection as data. As executable code, its level of integrity might be either high or low.

3.3 Integrity Model

The Biba model [8] describes an integrity policy that ensures the protection of high integrity entities (subjects and objects) from corruption. Such corruption can take different forms:

- A high-integrity subject reading or executing a low-integrity object (resulting in corruption of the subject's behavior).
- A low-integrity subject writing to high-integrity object (resulting in corruption of the high-integrity data).

Thus, a low integrity subject can always execute a high integrity program or read high integrity information, or write to low integrity objects, but a proper implementation of integrity policy will prevent the corruptions listed above. This can be expressed as follows.

Given the set S of active subjects and the set O of passive objects:

$$\begin{aligned}
&\forall\ s \in S, o \in O(\\
&\qquad currentaccess(s, o, read) \mid currentaccess(s, o, execute) \\
&\qquad \Rightarrow dominates(integrity(o), integrity(s)) \\
&\quad \& \\
&\qquad currentaccess(s, o, write) \\
&\qquad \Rightarrow dominates(integrity(s), integrity(o)))
\end{aligned}$$

In some systems, the confidentiality and integrity values for an object are combined into a single "access class" or label, as found in the GEMSOS model [30]. This model reflects the finding by Denning that all mandatory policies can be described using a lattice model where access classes are partially ordered [13]. Here, the separate confidentiality and integrity lattices are combined into a single "access class" lattice, and combined confidentiality/integrity access control decisions can then be based on comparisons of points (e.g., subject label and object label) on the combined lattice.

3.4 High Assurance Systems

In certain automated information system (AIS) environments, it is required that critical security policies be enforced with a high degree of assurance. The evolution of high-assurance systems can be seen to follow that of traditional computer architectures, from monolithic stand-alone machines [16, 37], to tightly-coupled distributed systems [36], to fully-distributed architectures, [42].

Early developments of high assurance systems demonstrated that they are both expensive to produce and are difficult to make compatible with commonly used (e.g., office productivity) applications. Examples include Blacker [42], various NATO multilevel projects, etc. These were custom-built systems that resulted in drawn-out programs and mutual incompatibility.

The "TCB subset" approach [28, 39] allows the allocation of separate policies, such as mandatory and discretionary access control, to separate and distributed components, and supports the separate evaluation of components, such that they can be composed into systems without necessarily re-evaluating all of the sub-components.

The balanced-assurance approach [29], as depicted in Figure 1, was introduced with the hope of reducing the expense of producing high-assurance systems, primarily by reducing the evaluation costs associated with less critical components. That is to say, low assurance components are combined with high assurance components to produce systems exhibiting high assurance with respect to the more critical of their security policies.

Similarly, the use of COTS components in government and military RFPs has become ubiquitous, even in requests for systems to handle sensitive information, at least partially in response to the problem of budgetary limitations in the modern economic era. Often, the use of COTS components is perceived to result in lower user training costs and an increase in interoperability between diversely developed systems.

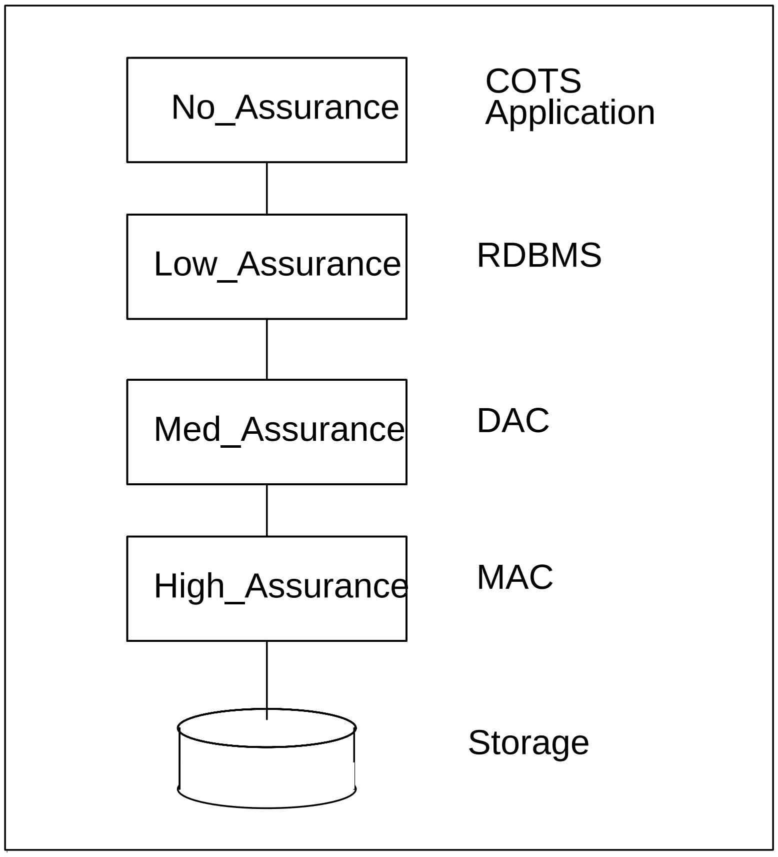

Figure 1: Balanced Assurance

The combination of the balanced assurance approach, COTS components, and fundamental multilevel security requirements motivated the development of various hybrid security architectures, discussed below.

4 Hybrid Security Architectures

In this section, we describe the *Hybrid Security* class of architectures, members of which are composed of a combination of low-assurance commercial components and specialized (e.g., high-assurance) multilevel components, and that are intended to enforce mandatory security policies while supporting unmodified COTS productivity applications. This class is characterized as architectures consisting of the following components:

- COTS terminals or workstations
- COTS user interfaces, applications and application servers
- Storage devices containing multiple levels of data
- Multilevel-management components

- TCB Extensions
- COTS network interconnections

where the components and their relationships are described as follows.

COTS Terminals or Workstations support client applications in relation to a local or physically-remote server component. Each terminal/workstation is configured to operate at a single user security level at a time (thus differentiating them from multilevel workstations or compartmented mode workstations). A characteristic of some COTS[1] Terminals and Workstations is that they are purged of all user data between user sessions at different security levels in order to achieve object reuse objectives, e.g. [27].

COTS User Interfaces, Applications and Application Servers COTS system applications provide useful user-level functionality, such as office productivity programs and database management services. Application servers may provide system application functionality to local or remote (e.g., "thin") clients. User interface applications translate user requests to the system application. System applications may be modified to be "multi-level aware" in their interactions with the Multilevel Management Component, such that they can return advisory markings to clients (user interface applications), or they can be purely single level [19].

Storage Devices may be storage disks or other media, the collection of which are capable of storing more than one security level of data. For some approaches, multiple levels of data are stored on the same disk, while in others there is one level per disk. Typically, the disks and disk controllers are commercial products, but we have seen proposals for multilevel disk controllers (see Multilevel Management Components, below).

Multilevel Management Components are "trusted" servers, operating systems, or device managers, and any attendant hardware. These components, sometimes called *reference validation mechanisms* (RVM), are interposed between clients or applications and storage media such that all communications and data flowing between the client workstation and the storage media passes through the Multilevel Management Component. The management software is responsible for enforcing mandatory and other policies with respect to data on the storage media. In a subset of the architectures under consideration, the management software will support the ability to "read down" from a user session at a high confidentiality level to data at a lower level(s), or write down from a session at a high integrity level.

Data at different security levels can be separated logically or physically. In logical approaches, multilevel managers ensure that requests for data are always bound to a particular security level and that policy enforcement is correctly applied to data access requests. Naturally, one form of multilevel manager is a full-blown high assurance trusted computing base incorporating a multilevel security kernel. Alternative managers could provide more limited functionality. In the case of physically separate single-level storage devices, a multilevel component

[1] In this report,"COTS" means a commercial product that has not been evaluated or has been evaluated below the class of B2/EAL5 (as there have been few, if any, commercial applications evaluated at B2 or higher, we consider this to be a conservative, non-exclusionary, definition).

might ensure that data access requests are shunted to the appropriate device. Clearly, not all architectures relying upon physical separation of storage media will be able to support read down functionality.

TCB Extensions. When components are separated from the Multilevel Management Component by network connections, there may be a "TCB extension"[2] associated with the component to reliably identify the component and its session level to the Multilevel Management Components, as well as to reliably manage end-to-end encryption, if that capability is provided.

Network Connections are logically or physically separate single-level "wires" between other components. In some cases they may multiplex transmissions at different security levels onto the same physical medium. Cryptography can be used to protect and separate information at various levels during transmission. Often different cryptographic keys are used for different levels.

4.1 Example Architectures

In this section we describe several instantiations of the hybrid security architecture. Note that we make no judgment or representation as to the adequacy of systems built to these architectures to enforce their declared or implied security policies.

A non-distributed instantiation of the model architecture is shown in Figure 2. 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 [14, 25], and "Purple Penelope" [33] (the latter includes a degenerate case of a Multilevel Management Component). A variation on this theme is the trusted VMM architecture, in which a separate version of the OS, in addition to the application and user interface, is created at each security level[22, 7], and multilevel management occurs below that in the VMM layer.

A simple distributed instantiation is shown in Figure 3. 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 may act to ensure 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 [6] (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 4. In this layout, there are logically separate single-level terminals (multiplexed onto one physical terminal by purging of state between session-level changes)

[2] A "Trusted Computing Base" (TCB) consists of the protection mechanisms of a computer system responsible for enforcing a security policy. A "TCB Extension"[10] is one or more such protection mechanisms that exist on a remote component, thus "extending" the functionality of the TCB.

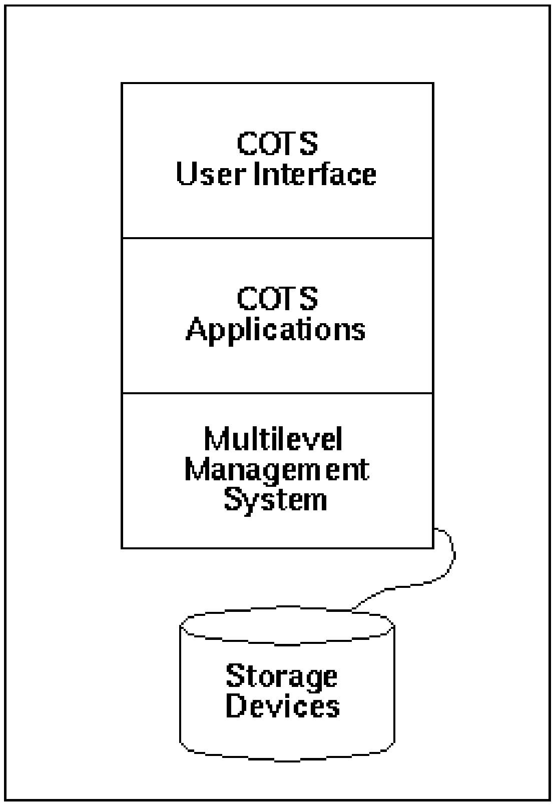

Figure 2: Single Process Architecture. (Network Connections are degenerate.)

connected via TCB extensions to multilevel-aware application server(s) and thence to the multilevel component. An example of this version of the architecture is that of the MLS LAN project [20].[3]

5 System Integrity 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.

Multilevel systems partition data into domains that are identified by security labels. Data of different sensitivities is stored in different partitions, such that some partitions are “more sensitive than,” “more trusted than,” or “dominate” other partitions. The dominance relation thus forms a lattice with respect to the labels/partitions (assuming the existence of labels for universal greatest lower bound, GLB, and universal least upper bound,

[3]In the MLS LAN design, the terminals would be equipped with a plug-in hardware component that is used to establish a trusted path from the user to a high assurance multilevel component on the network. The trusted path is used for user identification and authentication as well as session level negotiation. Once the session has been established, the workstation operates at a single, negotiated, security level [20]

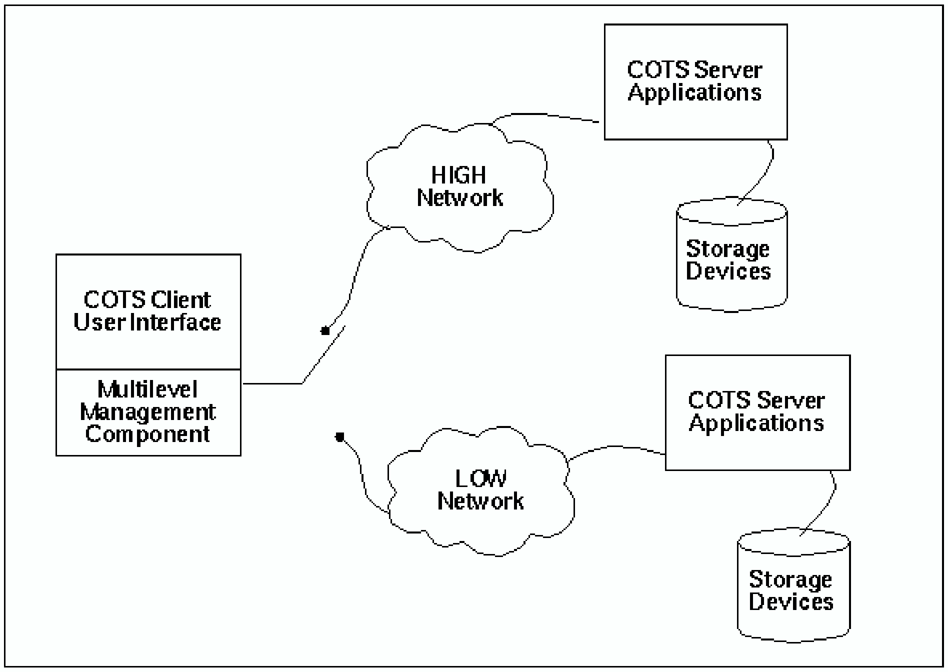

Figure 3: Switch-Based Architecture.

LUB). A reference validation mechanism (RVM, see "multilevel management component" in Figures 2, 3 and 4), mediates access to objects, controlling object creation, storage, access and I/O, thereby preventing data leakage across MAC partitions. For confidentiality 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 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, but not all, multilevel systems are designed to constrain flow between integrity domains.

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 domains and using the RVM to ensure that the security level of the user session matches the security level of the domain with which it is connected (e.g., Figure 3), or (2) using the RVM to logically separate the different data domains and to match the user session level to only the appropriate domain(s) (see Figures 2 and 4).

The central question in this paper is, "For what range of data can we trust such a multilevel 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.

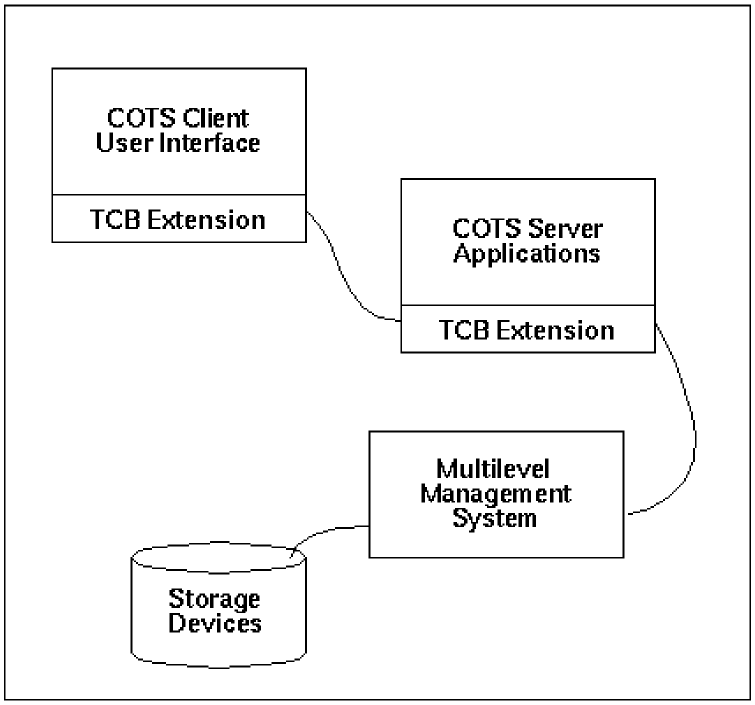

Figure 4: Distributed Multilevel Server Architecture

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, since the RVM will constrain the actions of the other components to not leak data across domains, 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. Then,

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

The "Yellow Book"[26] is an example of a scheme for assigning such 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.

5.2 Integrity Capacity

For integrity, on the other hand, *a system can be trusted to manage data (only) to the integrity limit of its least-trusted "data-modifying" component*. This is the "system integrity capacity." The reason this definition is different from the definition for system confidentiality capacity is that in addition to being concerned with the leakage of information across integrity domains, integrity enforcement has another semantic requirement, which is that the *allowed* (e.g., intra-domain) modifications must be the *correct*, intended, modifications. This reflects the fact that all *executable components* and code modules have an inherent or explicit integrity value relating to their faithfulness to ideal design objectives. So, we "trust" that a component will handle modification of objects correctly, (only) to its implicit or explicit level of integrity/assurance.

We will introduce C_{mod}, as the subset of components in system C that are able to modify user data. The integrity capacity, $i\_capacity$, of the system can be described as follows:

$$i\_capacity(C) \;=\; \underset{c\,\in\,C_{mod}}{\text{GLB}}\left(integrity(c)\right)$$

Proof Sketch[4]

There are two cases to examine regarding the definition of system integrity capacity: systems that do and systems that do not explicitly enforce an integrity policy based on integrity labels.

Axiom

A modified datum takes on the integrity of the GLB of the inherent integrities of the original datum and the data-modifying component.

$$\forall\, c \in C_{mod}\;,\; o \in O: \;\; modify(c, o) \Rightarrow integrity(o') = GLB(integrity(c), integrity(o))$$

For example, if a "certified" datum is modified by an "untrusted" code component, the modified datum becomes "untrusted," assuming that "certified" dominates "untrusted."

Case One

If the system does not enforce integrity, then it is the case that the system can take in data of any integrity level, including data of higher integrity than the inherent integrity of the overall system. The data is handled and managed by different system components, forming a $path$ of components between the user (or decision-making application, in the absence of a user) and the data store. We will call such components that have the capacity to modify or control the modification of user data, "data-modifying" components. From the Axiom, for each data-modifying component along the $path$, the data will take on an inherent integrity value equal to the integrity GLB of the data input to that component and of that component. By extension, no matter what the intitial integrity of a datum that enters such a system may be ($integrity(o)$), it will emerge from the system with a *de facto* integrity value no higher than the integrity of the least-trusted data-modifying component:

$$\forall c \in C_{mod}: \;\; dominates(integrity(c), integrity(o'))$$

Case Two

[4]A more formal presentation of this argument involves a mathematical representation of "path" and "data modifying component" based on the use of trace notation. For the sketch presented here, this level of detail is not critical.

If the system enforces integrity, when a component accesses an object for the purposes of modification (e.g., creation, deletion, append), the RVM enforces this rule: the integrity label of the subject (component) must dominate the integrity label of the object (no integrity "write up"). This rule is enforced for each data-modifying component in the system (alternately, the whole system could be labeled with the integrity GLB of all of the data-modifying components, and the rule enforced once upon entry of the object to the system, and reflected as the upper limit on the integrity label applied to objects leaving the system). Thus, the system will only be able to accept[5] objects with integrity labels no higher than (i.e., dominated by) the integrity of the least-trusted data-modifying component (again, leading to the definition of system integrity capacity):

$$accept(C, o) = \forall c \in C_{mod} : \quad dominates(integrity(c), integrity(o))$$

$\square$

For objects intentionally upgraded above the level of the either the component or the existing data, we would apply the logic of Case 1. An example of such an upgrade might be the transition of software from "development" to "production".

This leaves the question of correct integrity labeling of components, such that the system integrity capacity can be accurately enforced or inferred. Confidentiality and integrity labels of passive data objects may be assigned based on the data owner's perception of the object's sensitivity (e.g., harm caused by unauthorized disclosure or modification). As discussed, the confidentiality range of active components (e.g., code modules) can be assigned based on the guidance of the "Yellow Book." Unfortunately, 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. System developers and integrators must use their best judgment to assign integrity labels to code. Usually integrity labels, as well as confidentiality labels, are assigned by the system designer to maximize system functionality while being consistent with the principle of least privilege [34]. 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, cannot be accorded a high-integrity label – otherwise, data leaving such a system might have an inaccurately high integrity value, as reflected in its actual or inferred integrity label.

5.3 COTS Component Integrity

Hybrid security architecture COTS components are of particular interest with respect to correct integrity labeling. Recall that we define COTS components to have been either unevaluated with respect to security policy enforcement, or evaluated below Class B2/EAL5. In the security and evaluation community, B2/EAL5 is considered to be the dividing line between high and low assurance systems. Components evaluated below B2/EAL5 are "low assurance" (see, for example, [23] This is so for several reasons [27, 3]:

[5]We assume that an object is accepted by the system only if it is accepted by all of the components in C_{mod}.

- Weak developmental assurance, for example to ensure that Trojan horses 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.

- Little or no code analysis, and *no* examination of code for Trojan horses, after manufacture (i.e., during evaluation). There is no requirement for code correspondence[6] 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 Trojan horses are not inserted after manufacture. 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 an integrity label is an indication of how well a component's behavior corresponds to an ideal (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 as to how high of an integrity label can be associated with such code (see Appendix B). 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 are not significant; rather, it is significant to the enforcement of integrity policies in hybrid security architecture systems that site security managers/administrators, data owners, and other security policy "stakeholders" understand the integrity semantics of these labels, and that the integrity labels for their data objects have suitable dominance relationships to the integrity labels of the system components.

5.4 Integrity in Hybrid Security Architectures

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

- A system's integrity capacity is the GLB of the integrity of its data-modifying modules.

- COTS components are generally data-modifying. The general functionality of COTS applications such as word processing, spread sheet, presentation, time management, and database tools indicate that, to be useful, they are intended to modify, as well as read, data.

- COTS components are of "no assurance" or "low assurance" integrity.

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

- *Hybrid security architecture* systems use COTS components on the data “path”. By definition, the user interface and applications of *hybrid security architecture* systems are provided by COTS components, and all user data passes through these components.

- Therefore, the system integrity capacity of a *hybrid security architecture* system is generally no higher than “no assurance” or “low assurance,” the exception being when all of the COTS components in the *hybrid security architecture* system can be assured to not modify data.

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.

6 Conclusion and Discussion

We have shown that the integrity of a computer system’s code modules limits the data integrity “capacity” of the system. We have hypothesized that this information about system integrity capacity may not have been clearly disseminated in the past. The motivation for this paper is to record that information.

We have discussed why COTS components should not be attributed with integrity properties above a certain “low-assurance” level, and that *hybrid security architecture* systems should not be trusted with data whose integrity is above that level. An implication from this conclusion is that *hybrid security architecture* systems are not applicable 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, and

- The production of high assurance system components.

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

One might ask if high integrity is ever achievable. The answer is yes, but not with the “no assurance” COTS components as they are available today (viz, where COTS 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 hides data from modification by all but high-assurance components. Examples of the first are evident in systems intended to perform safety-critical functions such as avionics and certain medical systems[24]. An example of the second is a client-server system composed of high-assurance client and server components that encrypt their communication such that it is protected from modification during transit (e.g., via a Virtual-Private-Network-style connection).

References

[1] *Quality Systems – Model for Quality Assurance in Design, Development, Production Installation and Servicing* . Number ISO 9001. Geneva, 1994.

[2] *A Systems Engineering Capability Maturity Model, Version 1.1*. Number CMU/SEI-95-MM-003. Pittsburg, PA, 1995.

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

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

[5] 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.

[6] 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.

[7] S. R. Balmer and C. E. Irvine. Analysis of Terminal Server Architectures for Thin Clients in a High Assurance Network . In *Proc. National Information Systems Security Conference*, pages 192–202, Baltimore, MD, Ocbober 2000.

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

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

[10] S. Bryer-Joyner and S. Heller. Secure Local Area Network Services for a High-Assurance Multilevel Network. Master's thesis, Naval Postgraduate School, Monterey, CA, March 1999.

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

[12] 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.

[13] D. E. Denning. A Lattice Model of Secure Information Flow. *Communications of the A.C.M.*, 19(5):236–343, 1976.

[14] 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.

[15] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall, Englewood Cliffs, NJ, 1976.

[16] L. J. Fraim. Scomp: A solution to the multilevel security problem. *Computer*, pages 26–34, July 1983.

[17] M. Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold, New York, NY, 1988.

[18] C. Irvine and T. Levin. Recollections about data integrity limitations in hybrid security architectures. In *Presentation to the Workshop on Innovations in Strong Access Control*, Monterey, CA, To Appear 2000.

[19] C. E. Irvine, T. Acheson, and M. F. Thompson. Building Trust into a Multilevel File System. In *Proceedings 13th National Computer Security Conference*, pages 450–459, Washington, DC, October 1990.

[20] 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.

[21] C. E. Irvine and T. Levin. Recollections about Data Integrity Limitations in Hybrid Security Architectures. Technical Report NPS-CS-00-TBD, Naval Postgraduate School, Monterey, CA, To Appear 2000.

[22] 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.

[23] 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.

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

[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] 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.

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

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

[29] National Computer Security Center. *DRAFT Trusted Database Interpretation of the Trusted Computer System Evaluation Criteria*, unpublished, November 1988.

[30] National Computer Security Center. *Final Evaluation Report of Gemini Computers, Incorporated Gemini Trusted Network Processor, Version 1.01*, 28 June 1995.

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

[32] M. C. Paulk. A Comparison of ISO 9001 and the Capability Maturity Model for Software, Computer Security Technology Planning Study. Technical Report CMU/SEI-94-TR-12, Software Engineering Institute, Pittsburg, PA, July 1994.

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

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

[36] R. Schell, T. F. Tao, and M. Heckman. Designing the GEMSOS security kernel for security and performance. In *Proceedings 8th DoD/NBS Computer Security Conference*, pages 108–119, 1985.

[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] 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.

[39] W. R. Shockley and R. R. Schell. TCB Subsets for Incremental Evaluation. In *Proceedings Third AIAA Conference on Computer Security*, pages 131–139, December 1987.

[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.

[42] C. Weissman. BLACKER: Security for the DDN Examples of A1 Security Engineering Trades. In *Proceedings 1992 IEEE Symposium on Research in Security and Privacy*, pages 286–291. IEEE Computer Society Press, 1992.

A Production Techniques for High Integrity Components

A high-integrity component, one capable of performing its described function to a high degree of certainty, must be implemented properly and be free of subversive artifacts. That is to say, the component must not only do what it is supposed to do, it must not do anything else either. Various approaches exist for ensuring these characteristics, including: (1) post development testing; (2) abstract process certification such as the Capability Maturity Model [2, 32] and ISO 9001 [1]; and (3) rigorous engineering processes. There is ample evidence that testing alone is insufficient to ensure against malicious artifices [15]. The abstract certification approaches are not specific to producing high integrity components (although they can be used to manage a rigorous engineering approach), so they are not discussed further.

Under the managed engineering approach, the system hardware and software components are:

1. thoroughly documented with respect to functionality and correct use
2. critically tested with respect to documentation
3. critically examined in source form with respect to documentation
4. under strict configuration management throughout the product life-cycle
5. delivered to users with evidence of non-tampering, and with tools for verifying continuing integrity
6. produced with high assurance development tools[7]

This process appears to be sufficient to ensure the integrity of components to perform their function to the current state of the art. Note, that various government-sponsored security evaluation criteria [27, 3] award certifications to software systems for achieving increments toward these goals. [27] provides extensive supporting rationale for its requirements; the following is a brief overview of some of this rationale.

To understand the behavior of, and have confidence in the pedigree of COTS software, one must both test the software to validate that it behaves the way it is documented (see items 1 and 2) and examine its code in conjunction with the specifications to verify that its design supports its function and that it is not encoded to do anything else (items 1 and 3). Functional minimization as well as code modularity aid in the latter. Additional assurance against malicious artifacts is provided by the maintenance of strict configuration control, such that all changes to the software are accounted for and related to documentation during the lifetime of the software (items 1 and 4).

Finally, a mechanism must be in place to ensure that the delivery of software from the vendor to customers does not provide opportunities for the modification, and thus subversion, of the code (item 5). This mechanism can

[7]High assurance hardware components and development tools are generally considered to be beyond the current state of the art in the development of computer systems.

range from the use of couriers to various automated techniques. A robust, high integrity public key infrastructure can facilitate the trustworthy delivery of software. For example, certified public keys can be used to provide digital signatures so that both the source and the integrity of electronically transmitted software components can be verified.

Overall, these items provide assurance regarding component integrity from product design through creation and maintenance phases, as well as from manufacture through component delivery. Of course, in the end, the product must be used correctly (installed, configured, administrated and operated) to perform correctly, and the procedures for proper use must be clearly documented (see item 1, above), and rigorously followed. Likewise, a system must have a suitable security design to be able to protect itself and maintain its integrity during operation.

B 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 [4] are devoted to describing clandestine code which may be activated using undocumented keystrokes. Frequently this code merely displays an animated list of the software developers' names. Sometimes the effects are extremely elaborate as in the case of a flight simulator embedded in many 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[31]. Subsequent investigations revealed that instead of a trapdoor, the code contained nasty remarks about corporate competitors and well as violations of company coding standards[11]. 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. (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[9].

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 either the source code for Unix or the source code for the C-language compiler. The presence of this sort of trap door is speculative in *any* compiler and must be addressed through life-cycle assurance for any compiler chosen for high assurance system development.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library, Code 013 Naval Postgraduate School Monterey, CA 93943-5100	2
3.	Research Office, Code 09 Naval Postgraduate School Monterey, CA 93943-5138	1
4.	Mr. William Dawson Community Intelligence Office Washington, DC 20505	1
5.	Ms. Deborah M. Cooper PO Box 17753 Arlington, VA 22216	1
6.	Ms. Louise Davidson N643 Presidential Tower 2511 South Jefferson Davis Highway Arlington, VA 22202	1
7.	Dr. Michael Jacobs DDI National Security Agency 9800 Savage Road Fort George G. Meade, MD 20755-6577	1
8.	Dr. Cynthia E. Irvine Code CS/Ic Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5118	10