Data Integrity Limitations in Highly 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 while utilizing commercial-off-the-shelf (COTS) workstations and COTS productivity software applications. We show that a property of these architectures is that, while 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.

1 Introduction

Data integrity is defined as "the property that data has not been exposed to accidental or malicious alteration or destruction." [14]

A person who has *integrity* is identified as being one who acts based on a set of internalized principles that are founded on a system of ethics. A person is *reliable* if others perceive them as available and consistent. In contrast, high integrity information is information that can be relied upon as the basis for critical decisions. An example will illustrate what we mean by information integrity. A company may store a trade secret in its information system. This trade secret is used on a regular basis to manufacture the product. If the trade secret is subjected to malicious modification, the company's product will be affected. Some trade secrets are: formulas used by chemical companies, a food vendor's "secret sauce" or "classic" formula, and the closed source code of software vendors.

High integrity information may be used to support critical "intelligence" decisions. This may include both government intelligence as well as corporate intelligence. Highly reliable information can also be used to determine the course of diplomatic, military or corporate decisions. Another sector in which there is concern regarding information reliability and integrity is in the area of consumer information. This includes financial, educational, health and other records.

High assurance systems are intended to ensure the enforcement of policies to protect the confidentiality and integrity of information. The intent of this paper is to provide an overview of a subtle aspect of system security that is not often openly addressed by the developers of high assurance trusted systems: that is, a system can only be trusted to manage data whose integrity is at or below that of its weakest component. Some developers may admit that the problem we characterize here is something "we have always known about," yet for the consumers of these systems, the problem may not be fully appreciated. Our premise is that builders of systems designed to provide high assurance enforcement of non-discretionary security policies often 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 [6], it is implicit 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 review of concepts associated with confidentiality and integrity, and continues with a discussion of the challenges to integrity encountered in commercial productivity software. Section 3 follows with a description of a model architecture which we believe characterizes a wide variety of modern distributed secure systems. Several example systems are described. A description of integrity limitations in systems based on the model architecture is provided in Section 4.

Conclusions and discussion in Section 5 complete the paper.

2 Background

This section sets the context for the presentation of the model architecture and its attendant integrity limitations.

2.1 Confidentiality and Integrity

Confidentiality and integrity policies are often encountered in complex systems and are sometimes not clearly differentiated. 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. Classification may be 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 is needed.

A given piece of information will have a integrity value as well as a separate confidentiality value. This is the case whether or not the information has been explicitly associated with an integrity or confidentiality "label," as is done in "multilevel" systems. These confidentiality and integrity values apply to code as well as data objects.

The confidentiality value attributed to a datum may reflect the damage caused to the owner by unauthorized *disclosure of the information; whereas the integrity value may reflect the damage caused by unauthorized* modification of the information. For code, "integrity" carries the additional meaning that the code works the way it is supposed to, and performs no other, e.g. hidden, functions.

Is there an example of information that is high confidentiality and low reliability? Potentially critical information from an unreliable source might fall in this category. It may be held in strict confidentiality pending confirmation by a more reliable source.

Information that is high integrity but low confidentiality might be "selected, well established facts." For example, the Oxford English Dictionary is considered to be a highly reliable (and definitive) source for definitions of words used by English speaking peoples.

Low integrity and low confidentiality information might be the boasts of the children on the playground.

As noted earlier, high confidentiality, high integrity information could be a trade secret.

The Biba model [6] is intended to describe an integrity policy that ensures the protection of high integrity entities (viz, abstractions of active subjects and passive objects) from corruption. This corruption can occur by way of a high-integrity subject reading or executing low integrity data (thus corrupting the behavior of the subject), or by a low integrity subject writing to a high-integrity object (corrupting the value of the object). A proper implementation of the integrity policy will prevent these corruptions.

2.2 Integrity of COTS Components

There is a tension between integrity and productivity in both the development and the acquisition of commercial computer systems. For development, fast time-to-market product cycles are in conflict with the type of methodical development processes capable of yielding high-integrity products. For acquisition, the desire for feature-rich ubiquitously-available software functionality is in conflict with the need to for products that have been carefully built to ensure security policy enforcement.

Organizations find commercial software productivity suites highly seductive because of the resulting productivity enhancement of the workforce. These productivity suites may include word processing, spread sheet, presentation, time management, and database tools. However, commercial-off-the-shelf (COTS) components[1], both OS and application, are considered to be of low assurance.[2] In the absence of universally recognized integrity labels, we will designate low assurance software components to have an implicit integrity label of *low assurance*. In Section 2.3 we describe a number of techniques that support the production of high integrity components. In general, rapid production of software in a time-to-market-driven economy discourages the application of these techniques. Two detrimental effects enabled by low

[1] We define COTS components to be either unevaluated by independent third parties, or evaluated below Class B2/EAL5. [13, 3]

[2] The evaluation classes below Class B2 and EAL5 do not require either substantial configuration management or code inspection for malicious artifacts during evaluation, and components in these classes are considered to be "low assurance."[11]

integrity production techniques are incorrect software functionality and subversive software artifacts. As we will see, such low integrity software can have a negative effect on the data it handles.

There is clear evidence that subversion of commercial software is more common than generally perceived. Entire websites [4] are devoted to describing clandestine code which may be activated using undocumented keystrokes. Frequently this code merely provides an animated list of the software developers' names. Sometimes these animations 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 are just as easily possible.

An indication of the serious nature of the problem was indicated 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. Code investigations revealed that instead of a trapdoor[3], the code contained disparaging remarks about corporate competitors and well as violations of company coding standards. 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.)

In his Turing Prize Lecture, Ken Thompson described a trapdoor in an early version of the Unix operating system [18]. The cleverness of the artifice was evident in that the artifice was 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 own 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 existence of this such an artifact in commercial compilers remains a subject of speculation.

2.3 Production Techniques for High Integrity Systems

To produce a high integrity system, its components must be implemented properly and reflect lack of subversion. Various approaches exist for ensuring these characteristics. The primary approaches are (1) post development testing; (2) abstract process certification such as the Capability Maturity Model [2, 15] and ISO 9001 [1]; and (3) rigorous engineering processes [3, 13].

There is ample evidence that testing alone is insufficient to ensure against malicious artifices [8]. The abstract certification approaches are not specific to high assurance or high integrity (although they can be used to manage a rigorous engineering approach), so they are not discussed further. Under the managed engineering approach the system is:

1. thoroughly documented with respect to functionality and correct use
2. critically tested with respect to its documentation
3. critically examined in its source form with respect to its documentation
4. under strict configuration management during its life-cycle
5. delivered to users with evidence of non-tampering

We assert that 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 and examine its code in conjunction with the specifications to verify that it is not encoded to do anything else. Additional assurance against 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. This configuration control may encompass all software development tools and the development environment itself.

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. This mechanism can range from the use of couriers to automated techniques. A robust, high integrity public key infrastructure can facilitate the trustworthy delivery of software. 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.

Finally, the product must be used correctly (installed, configured, administrated and operated) to

[3] A trapdoor is an artifact that allows illicit entry into the system.

perform correctly, and the procedures for proper use must be clearly documented (see item 1, above), and rigorously followed. Likewise, the system must have a suitable security design to be able to protect itself and maintain its own integrity during operation.

3 Model Architecture Described

In this section, we describe a class of potentially high assurance multilevel architectures that support multilevel security and unmodified COTS productivity applications while utilizing (primarily) COTS components. The architectures in this class are characterized as distributed systems consisting of the following components:

- COTS terminals and workstations
- COTS user interfaces, applications and application servers
- COTS storage devices
- Multilevel management components
- TCB extensions
- COTS network interconnections

where the components and their relationships are described as follows.

Client terminals or workstations are logically- or physically-separate terminals or workstations each of which is configured to operate at a single user security level at a time (thus differentiating them from multilevel workstations or compartmented mode workstations). Each workstation executes COTS office productivity applications. A characteristic of the client terminals and workstations is that they are purged between user sessions at different security levels in order to achieve object reuse objectives, e.g. [13]. It is worth noting that the workstations may be equipped with a plug-in hardware component that can be used to establish a trusted path from the user to a high assurance multilevel component on the network. The trusted path would be used for user identification and authentication as well as session level (re)negotiation. The workstation operates at the selected, single, security level for the duration of a given session.

Storage devices may be storage disks or other media. More than one security level of data can be stored on these devices. For some approaches, multiple levels of information are stored on the same disk, while in others there is one level per disk.

Multilevel management components consist of trusted servers, operating systems, or device managers, and any attendant hardware. These components are topographically interposed between clients and storage media such that all communications and data flowing between the client workstation and the storage media must pass through the multilevel management component. The management software is responsible for enforcing mandatory policies with respect to data on storage devices.

Data at different security levels can be separated logically or physically. In logical approaches, multilevel device 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 device manager is a full-blown high assurance trusted computing base incorporating a reference validation mechanism. Alternative managers could provide more limited functionality. In the case of physically separate single level storage subsystems, a multilevel component might ensure that data access requests are shunted to the appropriate subsystem. Clearly, not all architectures relying upon physical separation of storage media will be able to support "read down" functionality (e.g., where a user session at a high security level can read data at a lower level) without gross covert backchannels.

When components are separated from the multilevel management component by network connections, there will generally need to be a "TCB extension" associated with the component to reliably identify the component and its session level to the multilevel management component.

COTS user interfaces, applications and application servers are either commercial products or freeware. 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 remote (e.g., "thin") clients. User interface applications translate user requests to the system application. System applications may be modified to be "multilevel 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 [9].

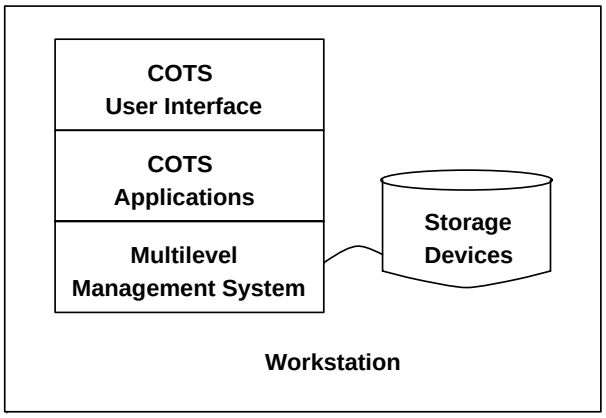

Figure 1: Single Process Architecture. (Network Connections are degenerate.)

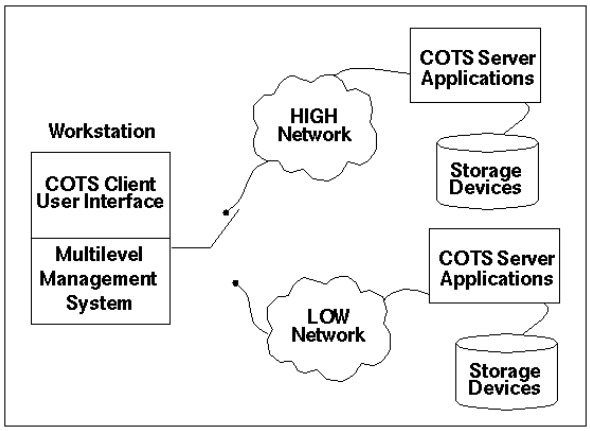

Figure 2: Switch-Based Architecture.

The last component of our architecture is the **COTS network connection**. These are logically or physically separate single level wires used for communication between other components. In some cases they may multiplex transmissions at different security levels on the same physical medium. In this case, cryptography may be used to transform information at various security levels to that of the transmission medium. Often different cryptographic keys are used for different levels.

3.1 Example Architectures

In this section we describe several examples of the model architecture. 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 1. In this layout, the architecture is process internal, where the lowest layer (e.g., "ring") of the process is a multilevel kernel, with an application server (e.g., multilevel-aware RDBMS) and application in higher layers. Examples of this version of the architecture are the Seaview project [7, 12], and "Purple Penelope" [16] (this includes a degenerate case of an multilevel management component).

A simple distributed instantiation is shown in Figure 2. Here, there are logically separate single-level workstations connected by a switch to different single level data-management subsystems. An example of this version of the architecture is that of the Starlight project [5], which is designed to allow low confidentiality to flow through the switch to high sessions. The third instantiation of the model architecture is shown in Figure 3. In this layout, there are logically separate single level terminals (multiplexed by purging of state between session-level changes) connected via TCB extensions to multilevel aware application server(s) and then to the multilevel component. An example of this version of the architecture is that of the NPS MLS LAN project [10].

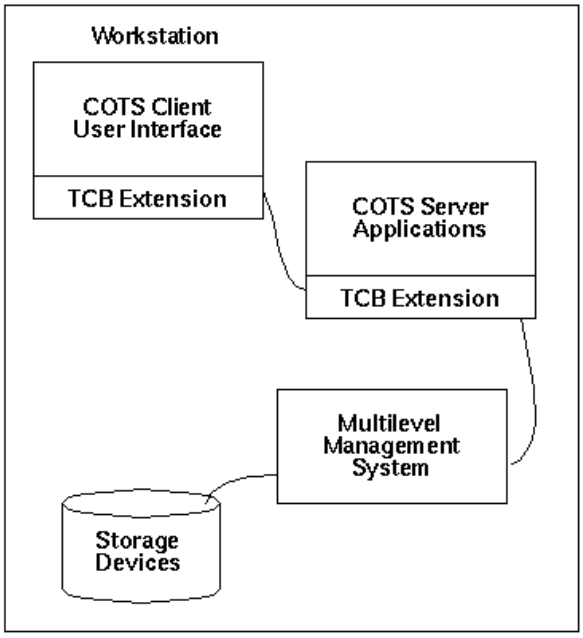

Figure 3: Distributed Multilevel Server Architecture

4 The Integrity Problem

In a system of components such as shown in Figures 1, 2 and 3, a centralized component that controls access to a set of protected objects can enforce a confidentiality policy on those objects with respect to the actions of other system components. Specifically, if the protected objects are partitioned into protection domains, this type of enforcement can ensure that data from high confidentiality domains does not exfiltrate to low confidentiality domains (other than, perhaps, through covert channels). However, this does not mean that high integrity data can be managed and protected by such a system. Rather, the integrity of data managed by a system can be no greater than the greatest lower bound (*GLB*)[4] of the integrity of all of the system components that may modify the object. This is the case whether or not the system enforces Biba integrity (or some other data or program [17] integrity policy).

If the system in question does not enforce an integrity policy, then high-integrity input data may be corrupted by low-integrity components in the system. This degradation is possible because the low integrity component may modify an object in an incorrect way, so modified objects take on the integrity of the greatest lower bound of the integrity of the modifying component and the original integrity of the object. Essentially, objects modified by low integrity components become low integrity objects.

On the other hand, if the system enforces an integrity policy correctly, then it will restrict the input of writable data to be only those objects whose integrity labels are less than or equal to the integrity of the least trusted system component that may modify the object. This is due to the integrity policy restriction that prevents integrity "write up" (i.e., a subject may not modify an object that has higher integrity).

Thus, in either case, the system is limited to maintaining data that is at or below the integrity level of its least trusted component. To illustrate this limitation, if a system consists of these components, all of which may modify user data:

- a *medium* integrity client application
- a *low* integrity COTS operating system
- a *high* integrity TCB Extension
- a *high* integrity TCB
- and a *medium* integrity application protocol server

Then the limit of integrity that the system can maintain is the *GLB* of (*medium, low, high, high, medium*) = *low*.

[4] We use greatest lower bound, rather than "minimum," because some integrity labels may not be comparable, meaning that the labels form a partial, rather than linear, ordering.

5 Conclusion and Discussion

We have shown that a system can only manage and protect data whose integrity is at or below that of its lowest integrity component. The hybrid security architecture (HSA) systems discussed here contain COTS components through which all user data flows and which have "low assurance" integrity. Therefore, HSA systems are only capable of maintaining "low assurance" data. These systems should not be used in automated information processing environments where there are high-integrity data requirements.

Many organizations have a "good enough" attitude regarding data security in multi-user environments, whereby the marketplace of commercially-available security solutions dictates the level of security provided to customer and corporate data. Losses associated with data security failures are considered part of the cost of doing business and are factored into customer pricing as well as corporate tax strategies. Nevertheless, each organization must make strategic decisions regarding which corporate and customer assets to expose to these shared environments. There is increasing competitive pressure for businesses to open their books to internet and other data-sharing environments. We expect that one impact of this paper will be to increase the level of understanding, and help quantify, the level of trust regarding data integrity that one should place in network/computer architectures that are based on COTS components.

One might ask if protection of high integrity data is ever achievable. The answer is yes, but only in architectures that prevent low-assurance components from modifying user data. An example is a client-server architecture where both client and server are of high integrity, and the data is protected (e.g., through carefully constructed protocols with high quality encryption) from modification when transiting low integrity portions of the network.

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.

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

[17] 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.

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