Security Risk Control of COTS-based Applications

Qun Zhong, Nigel Edwards
Networked Systems Department
HP Laboratories Bristol
HPL-97-108
September, 1997

E-mail: [qz,nje]@hplb.hpl.hp.com

commercial-off-the-shelf software, security, CMW, mandatory access control, discretionary access control

This paper introduces the CMW (Compartmented Mode Workstation) platform and its implementation on the HP-UX operating system HP-UX/CMW. It demonstrates a method of encapsulating untrusted COTS components without the need to check and verify the components' source code. By encapsulating COTS components on this secure operating system, we can control the system security while at the same time enhancing system reliability and take advantage of building the system with COTS components.

Security Risk Control of COTS-based Applications

Qun Zhong, Nigel Edwards

Hewlett Packard Laboratories, Bristol

Email: qz, nje@hplb.hpl.hp.com

Abstract:

This paper introduces the CMW (Compartmented Mode Workstation) platform and its implementation on the HP-UX operating system HP-UX/CMW. It demonstrates a method of encapsulating untrusted COTS components without the need to check and verify the components' source code. By encapsulating COTS components on this secure operating system, we can control the system security while at the same time enhancing system reliability and take advantage of building the system with COTS components.

1 Introduction

Nowadays, large scale information systems are increasingly built by using standard Commercial-Off-The-Shelf (COTS) components such as programs and linkable code libraries. Making use of such components can substantially reduce the complexity, time and cost in developing these systems: we are free to choose COTS components based on standard up-to-date technology.

Unfortunately, COTS components are usually developed by third parties and shipped as "black-box" products. Using them in your enterprise's information system can impose significant security and reliability risks. This problem becomes increasingly critical in today's network environment in which the Internet allows businesses to stay in touch with customers at a speed and convenience unmatched by other communication media. Exploiting this medium involves risking internal information used by the COTS components being leaked out through the globally connected network. For example, can you trust a third party's product to process your company's highly confidential financial data without sending the critical information back secretly? Can you trust a third party's product to be reliable and not corrupt the system on which it is running, or to unwittingly introduce a Trojan Horse?

Conventional operating systems do not provide the necessary facilities to confine the behaviour of a program without detailed knowledge of its internal structure. Most popular commercial operating systems have a security model based on Discretionary Access Control (DAC). This model puts the responsibility of maintaining security onto each individual user. Although this mechanism is very flexible, it is difficult to provide controlled access to a shared resource. Users who own resources have the discretion to assign what access permissions they wish. This means that they may impose very weak controls and unwittingly introduce security problems (e.g. global read and write on a Unix ".rhosts" file). This weakness makes it extremely difficult to control the security risks of a system that is built from a set of cooperating COTS components coming from different vendors.

A B-level [5] [6] certified operating system is a more suitable platform since it adds a set of security extensions, particularly Mandatory Access Control (MAC), not found in conventional operating systems. These security extensions make it possible to provide a restricted environment to encapsulate a system component according to its intended behaviour.

B-level certified (or higher) operating systems were typically developed and supplied for government

or military use. With the increasing need for security in the commercial sector, these operating systems are now finding wider areas of application. In this paper, we analyze the security risks posed by using COTS components. We provide an introduction to the security features of HP-UX/CMW, a B1-level version of HP-UX. We then describe how to make use of these features to design a security architecture for COTS components.

The solution described in the paper is derived from the highly successful Internet security product HP Presidium/VirtualVault [1] and research work on commercial security using CMW [2] [3] [4]. These examples demonstrate how to build secure systems using untrusted COTS components. By encapsulating the COTS components on a B-level operating system, we restrict the ability of each component to cause damage. Thus existing applications can be integrated with various COTS components without compromising system security. The B-level operating system is transparent to the end-user and COTS component, so minimal redevelopment is required.

2 Security Risks of COTS-based System

Since COTS components are developed by third parties independently, it is impossible to inspect the code to make sure that the component does not contain malicious code or that the code does not contain bugs that could corrupt the system under certain situations.

From the security viewpoint, we do not care what is going on inside the COTS components. We need be concerned only that the resources that it accesses are being used appropriately. Although there may be numerous potential causes of component misbehaviour, the types of damage caused by the misbehaviour are much more limited. Thus we focus on preventing the component causing damage if it behaves incorrectly.

If we regard COTS components as black boxes and study their external behaviour, then their ability to damage a system comes from either inappropriate access to resources or abuse of privileges they have been given. Further analysis results in the following list of security threats from unexpected behaviour of a component.

1. The component may access a resource in a way that it is not supposed. For example, a broken WWW Server writes to an HTML files it should only read and deliver (your home page is overwritten). This may also cause reliability problems to the whole system since one corrupted component may damage the resources used by other components and cause another component to generate incorrect results.
2. The component may access resources or services that it is not supposed to use. For example, an attacker can use Sendmail to read the /etc/passwd file and send it to an external system.
3. The component may abuse the privileges it has. For example, when attackers compromise and gain control of a component running with root (or "super-user") privilege, they can use the privileges to do whatever they wants to do.

3 Platform Requirements to Control Security Risks

As we have already seen in above section, there are two aspects to security risk:

- What system resources the component has access to;
- What privileges it has.

To confine the behaviour of the component, the minimum requirements are that we are able to specify and enforce: what resource are accessible in which way; what privileges should be given to which part of the component. The privileges given to components should be the minimum set needed to fulfill their function and no more: this is the principal of "least privilege". It should not be possible for the component or user to override these restrictions.

Our approach uses Mandatory Access Control (MAC) defined in the Orange Book [5]. This is based on a sound mathematical model of confining the usage of system resources to guarantee that the use of the system cannot breach the confidentiality of information. The concepts of "Compartment" and "Classification" can be used to specify the system resource requirement of COTS components logically. The operating system that implements this model has a kernel that enforces a non by-passable security policy.

4 Security Concepts of CMW

The Compartmented Mode Workstation (CMW) is an operating system with a set of security features defined in the Compartmented Mode Workstation Evaluation Criteria (CMWEC)[6]. CMWEC is a related but different evaluation criteria from the Orange Book [6]. CMWEC's criteria contain all the B1 security features as defined in Orange Book with some extra features. These security features are standard and independent to the operating system architecture so they can be integrated into various operating system platforms. The examples in this paper are implemented on HP-UX/CMW, which is a security enhanced HP-UX operating system previously supplied to government and military departments. Applications and shared libraries developed on HP-UX can be run without modification on HP-UX/CMW.

The security features in HP-UX/CMW include Discretionary Access Control (DAC) mechanisms based on security attributes such as identities (e.g. user IDs) found in a conventional operating system and Mandatory Access Control (MAC) mechanisms based on sensitivity levels built upon the concept of compartment and classification. The purpose of MAC is to enforce administration security policies. MAC and other related security mechanisms provide a platform that can impose the administrative access control policy on a process or a segment of code which is orthogonal to the application architecture and functionality. This ability provides the necessary tools to confine the behaviour of COTS components without the need to program the security policy into the components' source code.

4.1 Compartment, Classification and Mandatory Access Control

Sensitivity labels are a combination of hierarchical classification and non-hierarchical compartments. Classification represents a kind of ordered relationship while compartment represents a kind of container (set) relationships. The relationship of two sensitivity levels A and B is defined as:

- A is equal to B if:
  1. A's classification is the same as B's, and
  2. A's compartment sets are the same as B's.
- A dominates B if:
  1. A's Classification is greater than or equal to B's, and
  2. B's compartments are a subset of A's.

For example, if we have two compartment: inside (I) and outside (O) and two classification: unclassified (U) and confidential (C) (C greater than U). We get following domination relationship:

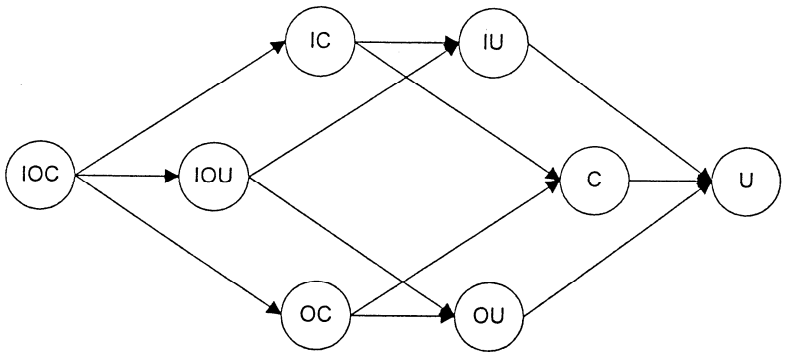

Figure 1 Domination Relationships

Mandatory Access Control is enforced by the operating system when a system operation happens. Every subject (such as a process or a user) and object (a system resource such as a file or device) in CMW is labeled with a sensitivity label. When a subject wants to perform a read/write operation on an object, the operating system decides whether to allow or reject the operation by comparing sensitivity labels of the subject and the object in addition to the normal discretionary access control check. The MAC rules CMW uses are: Subject A can read object B only if A's sensitivity label dominates or is equal to B's sensitivity label; Subject A can write object B only if A's sensitivity label equals B's sensitivity label. This rule is based on the lattice security model [7]. It guarantees the information flow cannot breach information confidentiality and integrity. The rule eliminates the potential security breach resulting from the ad-hoc method of explicitly specifying which component can access which resource.

Mandatory access control is the administration imposed access control. The sensitivity label of an object or the subject is defined in a "Trusted Computing Base" and is not changeable by users, unlike discretionary access control where whether an object is accessible to a particular subject is at the discretion of the owner of the object.

MAC provides an efficient way to achieve information separation that is critical in increasing the reliability and controlling the security risks of cooperating COTS components. By labeling system resources into different compartments and classifications, we can separate the resources used or shared by different components to avoid interference when one of the components misbehaves.

4.2 Privilege

In CMW there is no concept of an all-powerful super-user (e.g. root or administrator), rather the power of such a user is divided into approximately 50 privileges. Privileges are the trust tickets given to the subject to enable information flow across different levels and to do some other system operations. Along with the Discretionary Access Control, MAC label checking is enforced at the lowest system operation level, i.e., system read/write. It guarantees that the default information flow cannot destroy confidentiality and integrity of information. Any information flow between different security levels occurs in an auditable way with the necessary privileges. When a MAC/DAC check fails, the operating system checks to see whether the subject has the necessary privilege to override this check before deciding to reject the operation.

Beside read/write operations, the operating system also contains other system operations to be used by applications such as binding to a privileged TCP port. CMW associates a different privilege to

each of these system operations. To be able to use a particular system operation, a subject must possess the correct privilege.

By providing fine grained privileges, CMW lets us grant the subject the least privileges it needs to perform its task without being able to access system resources that it does not need.

4.3 Trusted Programming and Privilege Inheritance

On HP-UX CMW, trusted, security aware programs can be written by following the guidelines set out in [8]. Trusted programming let us fine tune the privileges to the instruction level so we can control the behaviour of a particular piece of code or function.

Trusted programs only raise privileges when they need to. Thus a program that needs to perform a certain system operation does not need to carry the privilege to do the operation the entire time even when it does not need to do the operation. For example, when we want to use a shared library function, we assign it only the least privileges it needs. If a library function misbehaves, it is less likely to lead to a serious problem if we carefully analyze the security risks and assigned it only the least privileges it needs before calling the function.

Controlled privilege inheritance between parent and child guarantees the safe transfer of privileges with great flexibility. For example, a child process cannot gain privileges it does not have accidentally by inheriting it from its parent process since privilege inheritance is not automatic. A parent process can choose to grant certain privileges to a child process and choose to withhold others. A parent process is not free to grant all privileges it has to its children: to grant certain privileges to its children the parent needs the "chsubjpriv" privilege. Thus a process does not necessarily get the same power as the user who started it. This is particularly important if the process is compromised and tries to start other processes.

5 How to Control the Security Risks

As in normal system development, when we want to enhance an application system's security or build an application that satisfies certain security requirements, we have to first analyze the system's security requirements. Then design and implement this security architecture. Provided with a suitable security platform, we can implement this security architecture while at the same time keeping the application's architecture intact.

This section looks at how to analyze the security risks and requirements. We provide some guidelines on how to design the security architecture to satisfy the requirements. Section 6 provides an example.

5.1 Security Risks/Requirements Analysis

The main activity of security risk analysis is the analysis of what resources and privileges particular components require. Depending on the particular security requirements and the understanding of the COTS component's internal architecture, the analysis can be carried out at different granularities. Generally speaking, if we have detailed knowledge of a component's structure, we can carry out risk analysis at finer granularity and gain finer control over the behaviour of the component. But if there is no requirement to have such a fine grained control, we can simply analyze the whole COTS component as one.

For some applications that need lots of dangerous privileges to run or need to access some critical resource, we need to have more understanding of their architecture. We can split the component into several parts according to with what and with whom it interacts. Then we can use appropriate

methods to encapsulate individual part to gain finer control of the component behaviour. An example is given in section 6.

5.1.1 Resource analysis

Resource analysis is essentially the study of what resources are needed by components in which way. It can be files, directories, hardware devices and network services. For example, a Web server needs to be able read HTML files it serve but should not be allowed to overwrite them. It should be able to write the audit log records but not to read them, etc. By defining what resources are required and how to use them, we aim at confining the component's resource access.

Resource analysis is carried out by studying the external behaviour of a component; often this behaviour is documented in the component's specification. There is no need to read through the code to determine it.

If the component need to access highly critical resources or need highly dangerous privileges to run, then there is the need to study its structure and decide which part of the component needs which resources so that we can have further control. For example, when the component needs to access highly critical resources, we have to ensure it only communicates with trusted components or goes through a security guard (proxy).

5.1.2 Privilege analysis

On CMW, the system's privileges are divided into a set of privileges that can be assigned to processes individually. We can study the component to determine what privileges it actually needs and just assign these privileges to it and nothing more. If the component needs highly dangerous privileges to run and we are not going to trust it to do so, we have to further study its structure to decide which part should be given which privilege. Again, this analysis can rely on the technical specification of the COTS component and does not require source code access.

If the technical specification is incomplete, the principal of assigning least privilege to the component means that the most likely outcome is that the component will have insufficient privileges. This is likely to mean that it will fail to execute, rather than execute in a way that damages the system.

5.2 Security Architecture to Control the Risks

The purpose of the security risk and requirement analysis is to enable us to design a security architecture according to a particular security requirement. The following are guidelines for designing the security architecture:

1. To prevent the COTS from gaining unnecessary or dangerous privileges, the following methods can be used:

   - When the component has to directly communicate with an untrusted world, such as the Internet, turn off the privileges that could be potentially harmful;
   - When the component has to be run in a privileged state to fulfill certain functionality, do not allow it to communicate with the untrusted world directly;
   - Provide a trusted link between the two sides of the application, i.e., unprivileged and privileged state.

2. By carefully labeling data with a suitable compartment and classification, we can thwart the threat of inappropriately accessing data. For example, give the read only data a sensitivity label that is

lower than the process, so the process can read it but cannot modify it.

3. To stop a broken component from interfering with other components, we can let other associated applications run in different compartments and provide a restricted communication path between them. The process that provides the path between two applications has the necessary privileges and can be programmed according to the trusted applications programming guide [8] to enable the process to perform the operations that cross compartments in a safe way.

6. Trusted Sendmail Proxy

Messaging services are one of the most popular components in network application systems. Sendmail is one of the most common programs that provide Internet mail service. However, Sendmail is also one of Internet attacker's favorite programs. Since it is big and complex, it is likely to contain lots of bugs that can be used to compromise it. Because it is a heavily privileged program, the result of compromising it can be very dangerous. In normal UNIX, it has to be run as the powerful "root" user. If it is compromised, the whole system could be under the control of the attackers.

Trusted Sendmail Proxy is a set of trusted programs to encapsulate the dangerous Sendmail application. It provides a sandbox to run Sendmail to reduce the damage caused by accidental or malicious use. Figure 2 shows the architecture of the Trusted Sendmail Proxy that provides a secure environment for privileged Sendmail.

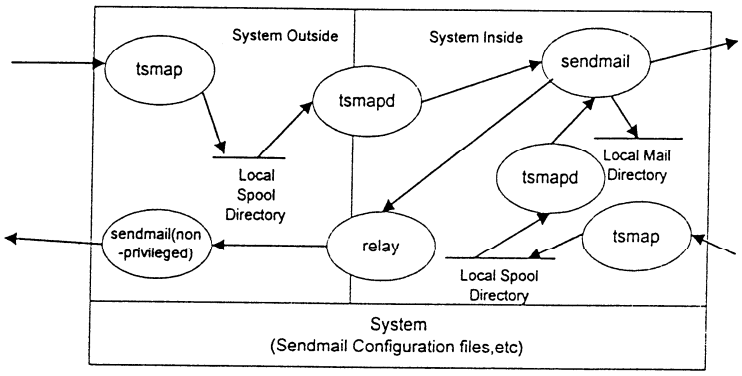

Figure 2: Architecture of Trusted Sendmail Proxy

6.1 Security Architecture Description

Since Sendmail is a heavily privileged program, the first thing we want to do is keep it away from directly interacting with outside world when in a privileged state. We use two compartments to divide the environment into inside and outside. The outside compartment is used for interacting with the Internet to collect the messages and for running unprivileged Sendmail to send messages to the Internet. The inside compartment is for running the Sendmail in a privileged state to deliver the messages locally.

Because there is no easy way of splitting Sendmail and making it accept messages without giving it privileges, we use TIS's method [9] to write a simple front end tsmap implementing SMTP only to collect the messages directed to this host and write the messages to a spool directory. This program

only needs two harmless privileges that let tsmap bind to the well-known port 25 and execute "chroot" to its spool directory so that tsmap can only access the spool directory to provide further protection.

Sendmail must have some potentially dangerous privileges when it has to call a mailer to deposit a message to a user mailbox or sending out messages in the name of the message sender. We put the privileged Sendmail in the inside compartment so that the host running Sendmail can act as the internal network's mailhost. Since we only give it the privileges to manipulate a user's mailbox but not the privileges that enable operation across the compartments, it cannot have any direct connection with the outside compartment.

We can see that CMW has an advantage over conventional operating systems here. In conventional operating systems, implementing similar functionality needs at least two physical machines.

We provide a small trusted program "tsmapd" as the trusted communication path to the privileged (system inside) Sendmail. This program reads the messages deposited in the spool directory by "tsmap", raises privileges that allow write operations across the compartments if necessary, and passes the messages to the privileged Sendmail to do the real delivery. Since some special formatted mail headers can trigger off some bugs in the mailers Sendmail use, we can also let "tsmapd" filter out known dangerous mail headers.

Sendmail in the inside compartment accepts the messages passed to it by either the inside or outside tsmapd. If the message should be delivered within the internal network, then this privileged Sendmail delivers it in the name of the message sender. If the message should be delivered to the outside network, the message should be passed to a trusted relay since this Sendmail does not have the privilege to do operations across the compartments. We modify the configuration file of this Sendmail to let it pass the "outbound" messages to our trusted relay in stead of the default built-in [IPC] mailer.

The small trusted relay program simply reads the messages and raises its privileges to pass the messages cross the border to the outside Sendmail.

The outside Sendmail performs complex operations to send out messages. It does not need any privileges to do so.

There are some common data files that both the inside and outside Sendmail want to read. We give these files a label that is dominated by both outside where the outside Sendmail is running and inside where the inside Sendmail is running so that both of them can read but not change the files.

6.2 Security Risk Analysis

If attackers from inside or outside manage to gain control of the daemon they are talking to (tsmap), they are unlikely to gain any profit since the program they are controlling only has the privilege to access the temporary undelivered messages in the spool directory. By scheduling the privileged tsmapd to run frequently, we can restrict this to a very few messages (possibly none). Besides, tsmap is a very small and simple program, so it can be inspected thoroughly.

The above architecture can also prevent the leakage of internal information and the Worm-like attacks even when the internal Sendmail is compromised. Mail messages carefully constructed by an attacker can trigger off bugs in Sendmail or in one of the mailers and fool it into executing some undesired code, as in the case of infamous Morris Internet Worm [10]. Since the inside Sendmail and its child processes are not able to open connections to the outside, the compromised Sendmail will not be able

to pull in the code from outside that does the real infection and damage (as in the Worm case). It will not be able to send out internal information either.

7. Current Status and Conclusions

This paper demonstrates that untrusted COTS components can be safely plugged into a system without major re-engineering, provided there is a suitable security platform. The method introduced in the paper is abstracted from HP's Internet security product HP Presidium VirtualVault and some other more recent results on encapsulating various applications to enhance application security and reliability. This demonstrates that Mandatory Access Control and related security mechanisms found in HP-UX/CMW can serve as a generic security platform to develop secure COTS-based applications.

Although the examples described have focused on HP-UX/CMW, the CMW features are operating system independent. For example, work is in progress to integrate the features into Windows NT.

MAC and related security mechanisms meet the diverse and large-scale commercial security requirement where risk control and application functionality are more important and practical than unbreakable security resulting from bug-free software. Sandboxing untrusted components on CMW allows us to have confidence in our application security since a broken component is not able to compromise our system security. It stops the "traditional" security competition between application developer and attacker on who find the bug first: the attacker to exploit it, or the developer to fix it. Bugs are only related to the reliability of the component not the security of the system. In addition, system reliability is enhanced since a faulty component is not be able to interfere with other components.

Reference

1 Hewlett-Packard Company, "HP Presidium/VirtualVault Internet Security Solution", 1996, available at http://www.hp.com/go/security/

2 Qun Zhong, "Providing Secure Environments for Untrusted Network Applications --- with case studies using VirtualVault and Trusted Sendmail Proxy", Proceedings of WET/ICE'97, IEEE Second International Workshop on Enterprise Security, MIT, June18-20, 1997

3 Nigel Edwards and Owen Rees, "High Security Web Servers and Gateways", Proceedings of the 6th WWW Conference, April 1997

4 C. I. Dalton and J. F. Griffin, "Applying Military Grade Security to the Internet", 8th Joint European Networking Conference, Edinburgh, U.K., 12-15 May 1997

5 Department of Defense Standard, "DoD Trusted Computer system Evaluation Criteria", DoD 5200.28-STD, Dec. 1985

6 Defense Intelligence Agency, "Compartmented Mode Workstation Evaluation Criteria Version 1(Final)", DDS-2600-6243-91, 1991

7 Ravi S. Sandhu, "Lattice-Based Access Control Models", IEEE Computer, Nov. 1993

8 Hewlett-Packard Company, "HP-UX/CMW Security Features Programmer's Guide", 1996

9 Trust Information Systems Inc. , "TIS Internet Firewall Toolkit Overview", available at http://www.tis.com/docs/products/fwtk/fwtkoverview.html, 1996

10 Jon A Rochlis and Mark W. Eichin, "With Microscope and Tweezers: the worm from MIT's perspective", CACM 32(6), June 1989, pp689-698.