Providing Secure Environments for Untrusted Network Applications

----With Case Studies using VirtualVault and Trusted Sendmail Proxy[1]

Qun Zhong

Hewlett Packard Laboratories, Bristol
Email: qz@hplb.hpl.hp.com

Abstract:

Bugs in network application program can be exploited to compromise the system on which the application is running. When running these applications in an unsafe environment such as the Internet, the security concerns raised are a significant barrier to electronic commerce. In addition, these application programs such as web servers, mailservers, etc., are usually too big and complex to be bug free; trying to build security directly into these applications has been proven very difficult.

The purpose of the paper is to demonstrate that Compartmented Mode Workstation is a suitable platform to provide a secure environment that can contain most existing network applications. We describe how to wrap these applications to reduce the potential for a security breach without the need to rewrite the application completely. By minimizing the effort of transferring unsafe application services to be reasonably secure, we are able to accelerate the process of electronic commerce.

1. Introduction

With global connection of Internet and easy access to various services on most hardware platforms, integrating existing IT application with Internet presents incredible commercial opportunities. However, deploying those applications also brings a major security challenge. Any process that lets customers into internal systems potentially opens a door to sensitive, mission-critical data and applications. Without the proper security guard, the danger coming from this door would keep many businesses from doing business on the Internet.

Various network security solutions provide different protection for different types of network attack [2]. However, security is an integrated system problem: any missing part can render the whole system vulnerable.

Today, most network security solutions do not guarantee an application server will not cause a security problem. Since most application servers are very big and complex, it is very difficult to insure that they will be bug-free and the bugs can be exploited to compromise the network system.

In this paper, we explain how to use Compartmented Mode Workstation, a B1-level operating system, to provide a safe environment for application servers. By encapsulating the existing application without rewriting it completely, we can gain the advantage of avoiding re-development and keep pace with application upgrades.

1.1 Background

Network security protocols such as SSL [9], SHTTP, etc., provide some protection over the communication channel between two ends. Firewall technology offers some protection by keeping out unwanted connections. However, most protection ends when the data reaches the server; they do not address what happens if the data itself triggers off bugs hidden in the server software or the server machine is compromised (for example an intruder gains a login). This can compromise the network system to which the server is attached, resulting in the leakage or damage of sensitive data on the server machine or contained in the internal network.

It is almost impossible to directly build security into every application. On one side, this would lead to rewriting the application in a simple, secure form. This is not feasible since most applications are very big and complex. They usually took great effort to develop. On another side, even if we manage to rewrite the program, we could notguarantee that the new program would be bug-free or safe. The sad story of Sendmail is an example of trying to build the security directly into the highly privileged program. New weakness/bugs of the program are identified continuously; new patches/versions have to be released and installed at the same fast pace in order to prevent a attacker from making use of them. It's not an

[1] Trusted Sendmail Proxy was carried out as part of the End-to-End Security (E2S) project in the European Ivth Framework Programme, IT RTD Project No. 20.563

easy task to install and configure an application every couple of months.

We have a dilemma: on one side, opening these applications to the customers on the Internet could bring brand new business opportunities and increase the competitiveness of the business. On another side, these applications are previously developed without security in mind and mainly used within business private networks, which are much safer than the Internet. Connecting these services to the Internet could raise substantial security problems that would be beyond the affordability of the business.

One of the techniques deployed is called sandboxing, a concept previously introduced in fault isolation [17]. In our context, it means running the vulnerable applications in a restricted environment thus confining the damage to the degree we can afford. In most of the situations, this means protecting the data from unwanted use by the application or/and prevent the attackers from gaining the privilege of the application when they break in.

However, conventional operating systems offer little choice in providing the restricted environment. It is very difficult to deploy the principle widely. Some work [10][16] make use of this principle to solve some specific problems. However, due to the limitation of the underlying operating system infrastructure, it's hardly suitable to use these methods to provide a general framework to solve the problem in this category. For example, [16] relies on physically separated hardware to provide information separation. [10] relies on specific operating system's process tracing facilities to trace the system call. There is still the powerful root user and the security checks are still at the user-level. There is still the possibilities that the attacker can make use of compromised privileged applications to alter the security policies and further his attacks.

1.2 Our Approach

We use HP-UX CMW (a B1 level operating system) to wrap untrusted applications. Operating systems certified as B1 or higher level provides a set of fine grained administrative security attributes and a set of operations to handle these attributes. Security checks are built into the Operating System kernel to guarantee maximum protection. Due to these features, it is able to separate the application structure from security structure, thus provide an ideal general platform for wrapping existing untrusted security-dumb applications.

To explain how CMW can be used to sandbox untrusted applications, we use two typical examples to serve as the case studies. One is the HP's Praesidium VirtualVault [11], which wraps the Web server and prevents it from unauthorized access/modification the data it uses. Another is Trusted Sendmail Proxy developed in HP Labs Bristol that prevents the heavily privileged yet vulnerable Sendmail from causing fatal damage to the system it running on.

The objective of the work is to serve as a general guideline to providing network application services safely by wrapping untrusted applications and to explore the advantages of introducing CMW (designed for military security) to solve the enterprise security problems in the commercial area. By minimizing the effort of developing the trusted application by wrapping the existing one, one can gain the advantage of avoiding redevelopment and keep pace with the application upgrade.

2. Security Concepts of CMW

The Compartmented Mode Workstation Evaluation Criteria (CMWEC) [6] was originally developed by Defense Information Agency for evaluating trusted systems used in the military and government. The CMW criteria is a different but related set of criteria to the more popular Trusted Computer Systems Evaluation Criteria (TCSEC or Orange Book) [7]. Under the TCSEC terms, CMW has all of the B1 level security features and includes some of B2 and B3 features. The HP-UX CMW is an HP-UX operating system with enhanced security features which satisfies CMWEC and has been passed the evaluation of B1 level. Applications developed on HP-UX can be run without modification on HP-UX CMW.

The most significant feature of B1 or higher level operating systems lies in its abstraction of administration security attributes in the computer system. In addition to the enhanced discretionary access control mechanisms based on the security attributes such as identities found in the conventional operating system, it also provides the mandatory access control mechanisms based on sensitivity levels built upon the concept of compartment and classification. The purpose of the MAC is to enforce administration security policies. The concepts of compartment and classification capture the essential comparable attributes of the real world and enable it to describe the security requirements found in the various applications naturally and gracefully.

We do not intent to cover all the CMW security features [12] here. This section only introduces some of the B1 level security features used in this paper.

2.1 Compartment, Classification and Mandatory Access Control

It is the Mandatory Access Control based on the sensitivity labels that distinguishes B1 or higher level operating systems from the conventional C and lower level operating systems. MAC is based on the administration policy concerned with information flow. Its mathematical model is Bell-LaPadula model [1]. It guarantees that no information flow will violate confidentiality and integrity (integrity is guaranteed in

CMW, not necessarily on B1 level systems).
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.

Mandatory Access Control is enforced by operating system when a system operation happens. Every subject (such as a process or a user) and object (such as a file or device) in a B level system 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 use is: Subject A can read object B only if A's sensitivity label dominates B's sensitivity label; Subject A can write object B only if A's sensitivity label equals B's sensitivity label.
Mandatory access control is the administration imposed access control. The sensitivity label of an object or the subject is defined in Trusted Computing Base and is not changeable by the user. It is different from the discretionary access control where whether an object is accessible to a particular subject is at the discretion of the owner of the object.

2.2 Privilege

Privileges are the trust tickets given to the subject to enable the information flow across different levels and to do some other system operations. MAC label checking is enforced at the lowest system operation level, i.e., system read/write operation level along with the Discretionary Access Control. It guarantees that the default information flow would not destroy the 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 by-pass 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 bind 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 relative privilege. By providing these fine grained privileges, there is a way to confine the subject in the minimum privileges it needs to finish its task and avoid the subject from abusing system resources that it does not need.

2.3 Trusted Programming

On HP-UX CMW, programs can be written in a trusted way by following the guidelines set out in [13].
The privilege inheritance between parent and child guarantees the safe transfer of the privileges with great flexibility. For example, a child process can not gain the privileges it does not have accidentally by inheriting it from parent process since privilege inheritance is not automatic. A parent process can have full control of its child process's behavior since a process holding a certain privilege can decide what privileges a child process can have, depending on the situation.
Trusted programs only raise privileges when they need to. This prevents the program that needs to perform certain system operation from carrying the privilege of doing the operation all the time even when it does not need to do the operation. If a bug is triggered off in a non-privileged area, it cannot lead to a serious problem. By keeping the privileged area of a program small and simple, we can put more effort into studying these pieces of code to keep them bug-free.

3. Using CMW to Wrap Untrusted Applications

Our objective is to protect ourselves from unwanted effects when an application server is compromised, for example, to prevent an intruder gaining root access to a system attached to the internal network by fooling Sendmail to hand this control to him.
CMW can be used to prevent the attacker from causing damage by manipulating vulnerable network application servers. In this section, we look at what the attacker is expecting from compromising the application and what we can do to prevent it. Then we use two typical network applications: Web server and Sendmail to illustrate how to apply the methods.

3.1 What we do not want to see from a compromised application

By analyzing the attacker's objectives and breaking them down into different categories, we are able to identify which part of the application is vulnerable to which types of attack. Then we are able to apply appropriate methods to strengthen it.
Common attacks' objectives are:

1. To gain the privileges of the application, for example root privilege, so that the attacker can do things he is not allowed to do with his current privileges.
2. To access the data the application uses in an

unauthorized way, e.g. modifying the HTML files the Web server delivers. One of the examples of how this could cause great damage is the "U.S. Department of Injustice" case [14].
3. Using the applications as a step to compromise another system that the attacker does not have the right to directly connect to. Since many applications need to cooperate with other applications, compromising one that the attacker can access can be used to attack another one that could not be reached before. For example, most Web servers rely on CGI scripts to provide information generated by other applications. Compromising the Web server could allow an outside attacker into the company internal network.

3.2 Using CMW to confine the attacker

To solve the above problems, we need two basic mechanisms: minimum privilege and information separation. Minimum privilege is needed when we want to reduce the function of a particular part of or the whole application to a certain degree. Information separation is needed when we want to prevent the server from accessing the data that it does not need or using data in an inappropriate way.
CMW is an ideal platform not only because it provides the mechanisms we need, but also because these mechanisms are built into the operating system kernel so that nobody can bypass them. Its Mandatory Access Control provides the information separation that prevents one process in one compartment from accessing data belonging to another compartment in an inappropriate way. Fine-grained privileges can let us control the behavior of the program without looking into the code.
Here are solutions to the above problems:
1. To prevent the attacker from gaining the privilege the application has, the following method can be used:
   - When the application has to talk to the outside world (Internet), turn off the privileges that could be potentially harmful;
   - When the application has to be run in a privileged state to fulfill its functionality, do not allow it to communicate with the outside world directly;
   - Provide a trusted link between the two sides of the application, i.e., unprivileged and privileged state.
2. By carefully labeling data the server uses in 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 can not modify it.
3. In order to stop the attacker from furthering his adventure even after he manages to compromise one application, we can let other associated applications run at different sensitivity levels and provide a strict communication path between them. The process that provides the path between two applications has the necessary privilege and can be programmed in a trusted way [13] to enable the process to perform the operations that cross the compartment in a safe way.

By analyzing the application according to when it has to interact with the unsafe world and when it is going to access certain types of data, we can decide how to break it up and use appropriate methods to encapsulate different parts. The following section uses two typical examples to show how to put the above guidelines into action.

3.3 Trusted Sendmail Proxy

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.
Sendmail has been one of the Internet attacker's favorite applications and the result can be very serious. Experts consider it one of the security nightmares. One thing that contributes to its fame is its long list of reported bugs [3]. Moreover, there are probably many other bugs that are still buried in Sendmail or not reported. The various techniques deployed in attacking Sendmail all make use of various existing bugs in Sendmail.
What makes Sendmail more attractive to attackers is that it is a heavily privileged application. It needs many privileges to run. In the normal UNIX, it has to be run as the powerful "root". Some of these privileges are very dangerous and not available to ordinary users. Once Sendmail is compromised and hands these privileges to the attacker, the whole system is probably comprised.
We cannot expect Sendmail to be bug free and we do not want to rewrite it in a trusted way [13] since it is too big and complex. The objective we aim at is to minimize the damage caused by exploiting it. In our context, this means that the damage is confined in the local host and no critical information on the local host can leak out. As indicated in the previous section, the strategy we adopt is providing a sandbox to run Sendmail. Figure 1 shows the architecture of the Trusted Sendmail Proxy that provides a secure environment for priviledged Sendmail.

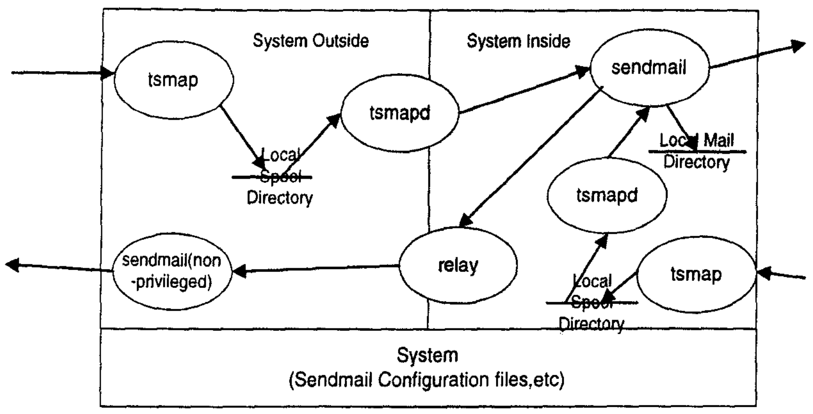

Figure 1: Architecture of Trusted Sendmail Proxy

3.3.1 Applying the Methods

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 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 [16] 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 one harmless privilege that let tsmap excute "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 the message to the 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 the users' mailbox but not the privileges that enable the 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 [16].

We provide a small trusted program "tsmapd" as the trusted communication passage to pass the messages to privileged Sendmail. This program reads the messages deposited in the spool directory by "tsmap", raises privileges that allow the write operations across the compartments if necessary, 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 "tsmtpd" filter out the known dangerous mail headers.

Sendmail in the inside compartment accepts the messages passed to it by either 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 put these files in the level 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.

3.3.2 Security Risk Analysis

If a attacker from inside or outside manages to gain the control of the daemon he is talking to (tsmap), he is unlikely to gain any profit since the program he is controlling only has the privilege to access the temporary undelivered messages in the spool directory.

The above architecture can also prevent the leakage of internal information and the Worm-like attacks even when the internal Sendmail is compromised. The mail messages carefully constructed by the 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 famous Morris Internet Worm [15]. Since 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.

3.4 HP Praesidium VirtualVault ---- Wrapping Web Server

VirtualVault is one of HP's Internet security products built on top of the HP CMW. VV provides secure access to the sensitive internal data for a commercial "off-the-shelf" Web server that directly interacts with the Internet. Serving static data alone is not enough in most situations. There is a need to connect other applications, for example, corporate databases, to the Web server.

A typical Web server contains too many complex functions and is too extensible to be trusted. Although the Web server itself does not need many privileges to run, it can be used as the stepping stone to attack the internal applications that interact with it

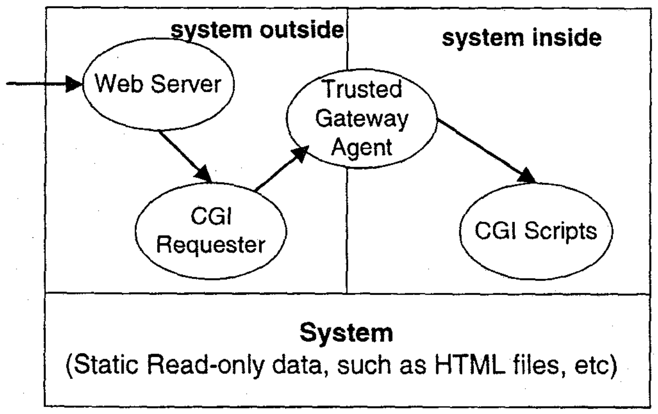

Figure 2: VirtualVault Architecture

The architecture we get is shown in Figure 2. We use two compartments, inside and outside, and one classification (system). The read-only information is labeled with the sensitivity label "system" without compartment. So the process running at sensitivity level "system outside" and "system inside" can read it but not modify it. The Trusted Gateway Agent provides a restricted secure path between the outside Web server and the inside applications.

3.4.1 Applying the Methods

The solution is to separate the applications and provide a restricted secure path between them according to the above methods. The point where Web server interacts with other applications is through the Common Gateway Interface. (This is the most popular interaction method Web servers use now. More efficient and capable secure interaction can be found in[8].) CGI simply spawns a child process to execute the program indicated in the URL and passes all the arguments to it. All we need to do is let the Web server run in the outside compartment so that it cannot access the internal sensitive resource; let all the CGI programs run in inside compartment so that they can cooperate with other internal applications. All the CGI scripts that can be used by the outside web server are registered with the Trusted Gateway Agent. All the outside web server's CGI requests are directed to the CGI Requester which passes it to the Trusted Gateway Agent. TGA validates whether the CGI request is a valid one. If it is, it then raises its privilege to invoke the CGI script in the inside compartment, passing out the result.

To protect the static read-only information, we should label it with a lower sensitivity label so that applications running in both inside and outside compartments can read it but not modify it.

3.4.2 Security Risk Analysis

The outside Web server only holds the privilege that enables it to bind to the privileged HTTP port. It is running at the level higher than the static HTML file's level, so it can read them but cannot modify them. If it is compromised, it can neither modify the static HTML files nor go into the internal system since it does not hold any other privileges at all.

Even if the data passed in from outside web server triggers off a bug in the inside CGI scripts or other application started by CGI, all damage is confined in the "system inside" compartment. Since all the "system inside" process cannot open connections to the outside network, the attacker is not able to directly control the broken process and make use of it.

4. Conclusion and Future Work

Due to fine-grained administrative security attributes and security operations, CMW provides an ideal platform to provide the tailored security management to meet various enterprise security policies. CMW meets the diverse and large scale commercial security requirement where risk control and application functionality are more important and practical than unbreakable security.
This paper demonstrates one of commercial application areas of CMW. Security-dumb applications can be safely put on the unsafe network without major re-engineering. It widens the service area of these applications. This is very attractive to a business that wants to make use of its existing IT infrastructure, or third party's products, to explore the new business opportunities on the Internet while at the same time protecting themselves from the various dangers it present.
CMW also show promise in other application areas, such as building new secure distributed object gateways [8], enhancing user authentication[5] and using existing single level applications to provide multi-level services [4].
By exploiting various commercial application areas of CMW, we aim to identify general security infrastructures for enterprise applications and to modify and extend CMW to build this platform.

5. Acknowledgments

The author would like to thank Nigel Edwards for his constructive suggestions and great help in preparing this paper. Also thanks Jonathan Griffin for his valuable inputs.

6. Reference

[1] D.E. Bell and L.J. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model", Mitre Corp. Report No. M74-244, Bedford, Mass., 1975

[2] W.R. Cheswick, S. M. Bellovin, "Firewalls and Internet Security - Repellimg the Wily Hacker", Addison-Wesley, 1994

[3] CERT advisory, ftp://ftp.cert.org/

[4] C. I. Dalton,"Providing Secure Multilevel Service using existing single level application", in preparation

[5] 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

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

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

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

[9] A.O. Freier, P. Karlton and P.C. Kocher, "The SSL protocol Version 3.0", http://home.netscape.com/eng/ssl3/ssl-toc.html

[10] Ian Goldberg, David Wagner, Randi Thomas and Eric A. Brewer, "A secure Environment for Untrusted Helper Applications --- Confining the Wily Hacker", Proceedings of 6th USENIX Security Symposium, 1996

[11] Hewlett-Packard, "HP Praesidium/VirtualVault Internet Security Solution", June 1996, available at http://www.hp.com/go/security/

[12]Hewlett-Packard, "HP-UX Compartmented Mode Workstation key security concepts", 1996

[13]Hewlett-Packard, "HP-UX CMW Security Features Programmer's Guide", 1996

[14] Noah Robischon, "Hacking Justice", http://cgi.pathfinder.com/netly/archive/netly/960819txt.html, 1996

[15] C. Schmidt and T. Darby, "The What, Why, and How of the 1988 Internet Worm", available at: http://www.mathcs.carleton.edu/students/darbyt/pages/worm.html

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

[17]Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham, "Efficient software-based fault isolation", In Proc. of the Symp. On Operating System Principles, 1993.