Operating System Support for the Sandbox Method and Its Application on Mobile Code Security

Qun Zhong, Nigel Edwards, Owen Rees
Networked Systems Department
HP Laboratories Bristol
HPL-97-153
December, 1997

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

mobile code security, sandbox, compartmented mode workstation, reference monitor

This paper discusses the problems arising when developing secure applications from both the security and the system engineering points of view. This paper demonstrates how Mandatory Access Control and the related privilege management mechanism can solve these problems by providing the non-bypassable security reference monitors to sandbox unsafe applications and by shifting the responsibilities of managing the security from the end-user to the security administrator. It introduces one of the available operating systems that provide these security features. It also describes how to use these features to solve the problems of implementing and using mobile code security through the example of a secure browser architecture we have implemented.

Internal Accession Date Only

Operating System Support for the Sandbox Method and Its Application on Mobile Code Security

Qun Zhong, Nigel Edwards, Owen Rees
Hewlett Packard Labs, Bristol
Filton Road, Stoke Gifford
Bristol, BS12 6QZ
< qz, nje,ore@hplb.hpl.hp.com>

Abstract

This paper discusses the problems arising when developing secure applications from both the security and the system engineering points of view. This paper demonstrates how Mandatory Access Control and the related privilege management mechanism can solve these problems by providing the non-bypassable security reference monitors to sandbox unsafe applications and by shifting the responsibilities of managing the security from the end-user to the security administrator. It introduces one of the available operating systems that provide these security features. It also describes how to use these features to solve the problems of implementing and using mobile code security through the example of a secure browser architecture we have implemented.

1 Introduction

Global connectivity introduced by the Internet provides exciting opportunities for sharing resources on a large scale. Mobile code technology allows us to utilize these resources and collaborate with each other efficiently. With this technology, we can automatically load programs from the network and run them on heterogeneous platforms. This means that it is possible to build large applications consisting of dynamically changing and cooperating components.

However, as well as making it feasible to build large scale distributed applications, the Internet and Mobile Code technology also expose us to various attacks from the unsafe public network. Not only can malicious mobile code attack the local host, but it can also bypass corporate firewalls to attack the internal network. Since almost all of our current network security products such as firewalls are not designed to resist attacks from the internal network, the security risk of using mobile code is significant.

The exciting potential of this technology has resulted in lots of research work in the security area. However, most of this work focuses on the security issues alone. It is difficult to convert the results to the real business applications for two reasons.

Firstly, using them requires substantial knowledge of security. The end-user also has to spend great effort to configure and maintain them. It is difficult for a security naïve end user to correctly configure and activate them. Additionally, these users are more likely to be subject to so-called social engineering attacks since they are less aware of the security implication of a request made by the application than the experienced and well informed security experts.

Secondly, it is difficult to guarantee that the final system will remain secure due to the bugs introduced in the implementation. The experience of Java security has shown that many security problems reported are not the problems of the Java security model itself [1] [2]. These problems not only come from bugs in the implementation of the security mechanism but also come from the programs that directly use the Java Virtual Machine(JVM), since the security check built into JVM can be by passed.

Unfortunately, it is almost impossible to eliminate bugs from any non-trivial engineering projects. What is feasible is to mitigate the damage caused by implementation bugs and malicious code. The sandbox method, a concept previously introduced in fault isolation [3], is deployed to meet the requirement. By providing a restricted environment to execute the unsafe code and confine its behaviour in a suitable way, it is possible to reduce the security breach to an acceptable degree. In addition, this method can accommodate errors introduced in the application development.

Through the concrete example of the vaulted browser we have implemented, this paper demonstrates how to use operating systems with Mandatory Access Control (MAC) and privilege management security features to support the sandbox concept. More importantly, we suggest that only through the operating systems with these security features, is it possible to provide a non-bypassable restricted execution environment.

The vaulted browser architecture described in this paper adopts several specialized application level reference monitors to control the resource access instead of building one big and complex reference monitor. This feature reduces the size and complexity of each reference monitor so the code of the reference monitor can be thoroughly studied.

By shifting the security configuration and activation of the vaulted browser architecture to the security administrator, the system usability and security can be greatly enhanced. This ability allows the security policies to be enforced consistently across the enterprise network. Consequently, the risk of inconsistent security policy can be greatly reduced. Additionally, this ability also liberates the security naïve end-users from managing the mobile code security themselves, and thus, enhances the system usability and reduces the risk of social engineering attacks.

2 Mandatory Access Control and Privilege Management

This section introduces Mandatory Access Control (MAC) and Privilege Management. MAC is based on the lattice security model [4]. It was developed and used to support information flow control. Its purpose is to enforce administration security policies. However, Mandatory Access Control alone is not sufficient to confine the behaviour of an application while at the same time provides the flexibility that the application needs to function. We need both MAC and the related privilege management mechanism to provide what we need to support the sandbox method from the bottom-up.

Operating systems providing these additional features are typically developed to meet or exceed the B level trusted operating system defined in the Trusted Computer

System Evaluation Criteria (TCSEC) [5]. HP-UX/CMW is one of these operating systems. It was developed to meet the Compartmented Mode Workstation Evaluation Criteria (CMWEC)[6]. CMWEC is a related but different evaluation criteria from TCSEC. CMWEC's criteria contain all the B1 security features as defined in TCSEC 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 descriptions of these security features and the vaulted browser we implemented are based on HP-UX/CMW, which is a security enhanced HP-UX operating system previously supplied to government and military departments. Most applications and shared libraries developed on HP-UX can be run without modification on HP-UX/CMW.

2.1 Mandatory Access Control

Mandatory Access Control is the administration imposed access control. It is enforced by the operating system through comparing the sensitivity labels of the subject (such as a process or a user) and the object (a system resource such as a file or device) of a system operation. This security check is in addition to the normal Discretionary Access Control (DAC) check that is available on most of the operating system today. These MAC labels are defined by the system administrator and are not changeable by the users who do not have certain privilege.

Sensitivity labels are a combination of hierarchical classification and non-hierarchical compartments. 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 compartments: inside (I) and outside (O) and two classifications: unclassified (U) and confidential (C) (C greater than U). We get following dominance relationship between the possible sensitivity labels:

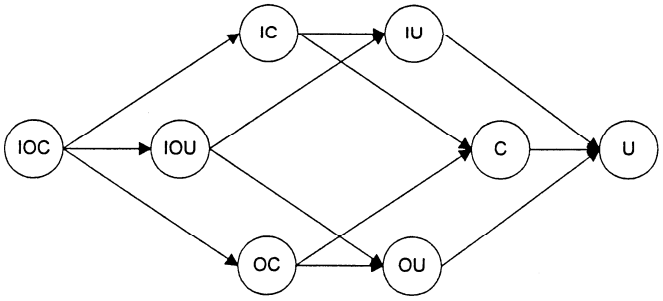

Figure 1 Dominance Relationships

For example, IC dominates IU and C as well as itself. But IC does not dominate OU.

The MAC rules CMW used are more restrictive than the rules specified by TCSEC: 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. It guarantees the information flow cannot breach both information confidentiality and integrity.

MAC provides an efficient way to achieve information separation that is critical in sandbox model. By labeling processes and system resources into different compartments and classifications, we can separate the subjects and the resources that will otherwise be achieved through expensive hardware separation. As a result, every non-default resource access has to go through a proxy that has the necessary privileges to overwrite the MAC rules. This provides excellent support to build the non-bypassable security reference monitors of the resources we want to protect.

2.2 Privileges and Privilege Management [7]

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, which can be grouped into three sets. These privileges are the trust tickets given to the subject to enable them to perform certain sensitive system operations. For example, 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. One set of privileges is used to override MAC rules. For example, "allowmacread" privilege enables a process to read data of different sensitivity label. Another set of privileges is used to override DAC check. Others are used to enable various privileged system operations such as binding to a privileged network port. Any operations that need privilege to perform are subject to system audit.

CMW also has a well-designed privilege management mechanism to guarantee the safe transfer of privileges with great flexibility. This mechanism greatly reduces the damage caused by compromised privileged processes in following ways. Firstly, the privileges of the users and executable files, which are assigned by the system administrator, are safely transferred to the processes. For example, if a user does not have necessary authorization that is assigned by the system administrator, the processes he or she started will not get the privileges the executable file has. Secondly, the privilege inheritance between parent and child processes is not automatic. As a consequence, even if the process is compromised and spawns malicious child processes, these child processes will not acquire the privileges their parent processes have. Finally, a program is able to check whether its parent process is a "trusted process", which means this parent process should hold a certain privilege at the time it started the child process. This ability lets the child process decide whether it should carry on or not. Consequently, a compromised application is prevented from executing the privileged executable file the user is authorized.

In addition, "trusted programming style" as set out in [7] employs the concept of "privilege bracketing". This style requires the programmer to raise the necessary privileges just before the sensitive system operation and remove them as soon as it is not needed. Programs written in this style are more reliable as the programmer can

browser and hence the mobile code it downloads and runs are isolated by labeling them with "System Middle".

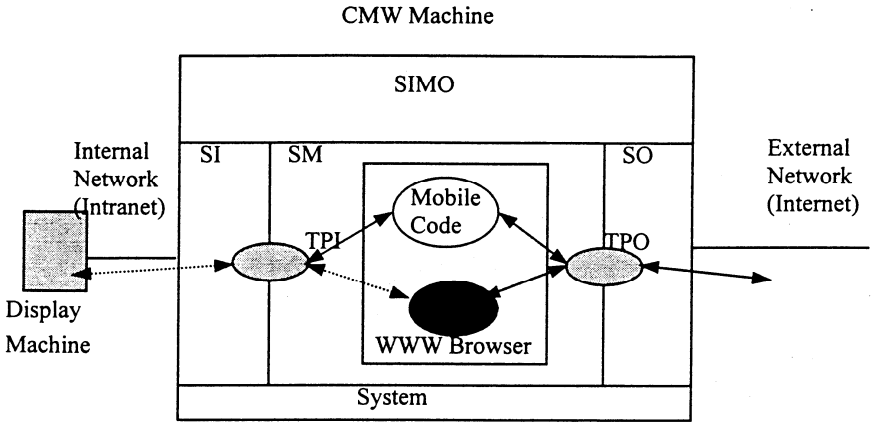

Figure 2

The browser's executable file, the files, directories and other resources that the browser read but not write such as the configuration files are given the label "System". The result is that these resources are protected by the Mandatory Access Control so that users, a broken browser or malicious mobile code cannot bypass the security administration by overwriting them with their own copies of these files. Other files that need to be both read and written by the browser, such as bookmark file, history files and cache, are labelled as "System Middle".

The browser and all the mobile code it executes in the System Middle do not have any privileges so that they cannot access any resource labelled with different compartments unless it requests them through a proxy. Therefore, the behaviour of the browser and mobile code is encapsulated in the "System Middle" compartment and the local resource access control is enforced by the operating system.

TPO acts as the Reference Monitor to the network resources. It enforces the network access control according to the policies set up by the system security administrator. It is a small program that has the necessary privileges that allow it to pass data between different sensitivity labels. The browser's Socks [7] proxy is configured as TPO to enable the browser and the mobile code to gain access to the network resources.

TPI acts as the Reference Monitor of the display resources. It has the privileges that all it to relay all the display messages to the display servers labelled as "system inside" using X protocol. These display servers are workstations running X servers, for example, Xterms or WindowsNT workstations running Xserver emulators. If the users are not working on the CMW machine, they can still browse the Internet safely since the display messages is relayed by TPI to reach the user's machine.

In a more generalized form, the vaulted browser takes following form:

pay more attention to study the privileged code segments to make sure bugs would not be likely in these areas. As a result, the chance of the failures happening in the privileged area will be substantially reduced.

By providing fine-grained privileges, CMW lets us grant the subject the least privileges it needs to perform its task. As a result, we can make use of this ability to confine the behaviour of the subject. For example, if we do not want a process to access certain resources but still be able to perform certain sensitive system operations, labeling the process and the resources with sensitivity labels that do not dominate each other and withhold the MAC privileges from the process will achieve the goal.

With the privilege management and trusted programming, we can program our critical trusted programs securely and reliably. The privileged trusted programs that implement the vaulted browser architecture described in section 3 make use of these security facilities and follow this programming style. They check whether they are started by the authorized parent process with proper privileges. This enables the small trusted programs that act as the reference monitors to be tamperproof.

3 Security Architecture of Vaulted Browser

This section describes how to make use of MAC and privilege management to provide a restricted executing environment for the WWW browser and the mobile code by preventing them from pretending to be local trusted processes and forcing them to pass through the reference monitors to access critical system resources. We will describe the security architecture first. Then we will describe an implementation of this architecture on HP-UX/CMW operating system. Finally, we will describe how this architecture can prevent various attacks on Java and the extra advantages it provides.

The prototype is implemented on HP-UX/CMW 10.09 with Netscape 3.0. However, this security architecture can be implemented on any operating systems provided they have similar MAC and privilege management with any browser that supports Socks [8] protocol.

3.1 The security architecture

The first goal we want to achieve is to prevent any process or thread started by the browser from pretending to be local trusted code. Therefore, as soon as the data for the browser, probably containing mobile code, arrives at the local host, we label them. The operating system guarantees that this label is not changeable by anyone who does not have the privilege, so the subsequent security decisions can be reliably based on this label. The second goal we want to achieve is to force the processes to go through the security check before they can gain access to the various resources. This is achieved by labeling all the resources we want to protect with different labels.

Figure 2 is the security architecture of the Vaulted Browser in its simplest form. We use one classification, "system", and three compartments: Inside, Middle and Outside, to label the resources and processes. All the hosts in the Intranet are labeled with "System Inside". All the hosts on the Internet are labeled with "System Outside". The

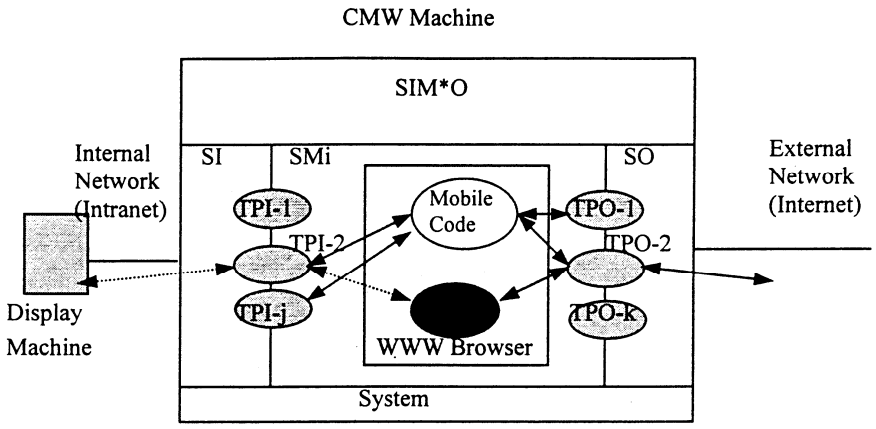

Figure 3

Where there are multiple middle compartments: SM1, SM2, ... , SMn. Each middle compartment represents a browser and mobile code that can access a particular set of resources. Different middle compartments can have different sets of trusted proxies that allow the mobile code to access different services. For example, in the above figure, TPI-1 can be used to provide limited access to CORBA services in the internal network using IIOP [9].

3.2 The Implementation of the Architecture

The implementation of the above architecture consists of four components: Trusted Browser Front End (TBFE), Trusted Proxy Inside (TPI), Trusted Proxy Outside (TPO) and a browser. TBFE acts as a single point-of-entry to the vaulted browser. It also terminates the whole process group when any single member terminates. TPI is a display proxy. It manages the interaction between the real browser running in the middle compartment and the display servers. It enforces the access control policy to the display resources. TPO is a network access control proxy. It's a modified Socks server that enforces the network access control. Other than the browser, which is not given any privileges, all the other three components are small and privileged programs that are written in trusted programming style [7].

Figure 4 shows the relationship between these four components and the communication protocols between them.

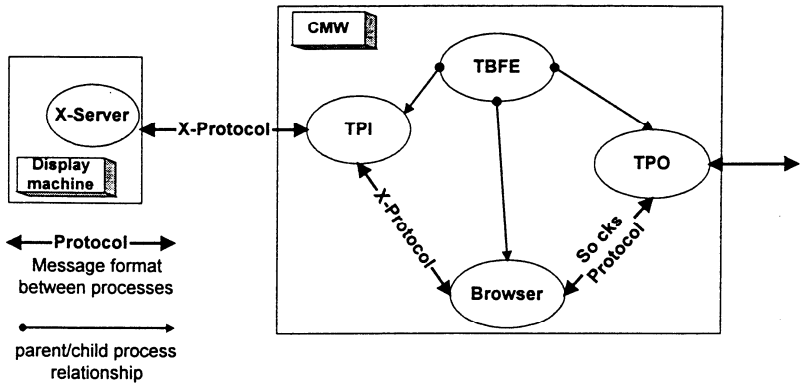

Figure 4 Processes and the messages between them within the display machine and CMW

TBFE is started by the user who wishes to use the browser. It has the privileges that allow it to start the browser and proxies in the correct compartments and pass them the correct parameters. TBFE then waits for one of the child processes it spawned to terminate. Usually this is the browser when the user finishes using it. Then TBFE sends exit signals to its other child processes and exits cleanly.

TBFE can be started remotely by users that are not connected to the CMW machine provided they have accounts on the CMW machine, they are authorized to use the vaulted browser system and a X-server is running on their local machine. In addition, a script that starts TBFE remotely, for example, by making use of remote execution functions provided by UNIX such as 'remsh' or 'rexec', should be installed on their machine.

When TPI is started by TBFE in System Inside, it waits for the connections from the browser running in System Middle. When TPI receives a request for display, it evaluates the request and makes a connection to the real display server the user is using. If this connection is successful, it then pumps the messages between System Middle and System Inside. Optionally, some X message filtering can be performed here to prevent suspicious X-messages generated by the mobile code getting through.

When the TPO is started by the TBFE in the System Outside compartment, it waits for the connections coming from System Middle. When a connection request arrives, the TPO evaluates the request based on the destination and the label of the request to decide whether the connection is allowed. If the connection is not allowed, it then drops the connection. Otherwise, it will make the connection to the destination and raise the necessary privileges to pass the data between the different sensitivity labels.

3.3 The Benefit and Security Analysis of the architecture

By providing the resource separation through MAC labels and dividing the whole system's reference monitor into several specialised reference monitors, this architecture can protect us from the attacks through application and JVM implementation errors and provides the potential to control the resource specific

attacks centrally. It also frees end users from managing and maintaining security themselves.

3.3.1 Security Analysis of the Architecture

The security of the architecture relies on the security of underlying operating system. However, the operating system is the only thing this architecture depends on. Directly building security mechanisms on top of operating system without any support from it also depends on the operating system's security. In addition, this method depends on the security of the applications that use these security mechanisms as well. As it is very difficult to be sure that these applications do not contain bugs, it is difficult to guarantee that compromised application can bypass these security mechanisms and access resources via native operating system directly.

The degree of dependency to the operating system security is smaller than security mechanisms without MAC and privilege management support. Unlike conventional operating system, it is easy to achieve information separation on operating systems with MAC. As many operating system attacks are through the bugs in privileged system programs, keeping the unnecessary system programs away from processes in middle compartments will reduce the security risk.

The architecture satisfies the three design requirements of reference monitors [4]. Firstly, the configuration files of these reference monitors are labeled with "system" which means that they are not changeable by the end-user or malicious mobile code. In addition, these reference monitors will check whether the process started them has the necessary authorization. These mean that these reference monitors are tamper proof since only the authorized process will be able to start them and they will be started with authorized configuration files. Secondly, the reference monitors such as TPI and TPO are not by-passable because only though them the resources labelled with another sensitivity label can be reached. Finally, as their tasks are mostly checking the sensitivity labels to see whether the data flow is allowed or not and not processing the data itself, they are very small can be thoroughly tested and analyzed.

Since all the access control checks are based on the immutable sensitivity labels and nothing else, there is no way that a piece of code can gain any excess privileges or access resources by bugs in applications and JVM. As a consequence, the attacks such as "Slash and Burn" [2], which allow the mobile code to pretend to be the local trusted code by exploiting the bugs or design flaws in the browser and JVM, will not be able to access the extra system resources. Since mobile code does not have any privilege, it cannot change its own label to let the TPI and TPO grant it extra resource access. Even though they can break the security enforced by Java, they are still confined to the "system middle" and able to cause only very limited damage to the resources labelled as "system middle". Attacks such as "Applets Running Wild", "Casting Caution to the Wind" and "You're not my Type" [2], which penetrate the system through exploiting type confusion and the security inconsistency in various pieces of Java security "prongs", will not collapse the whole system due to the same reason.

As all the access to a resource has to pass its reference monitor, this reference monitors can be further enhanced to provide extra resource specific protection when

needed. For example, The TPI in our architecture can be enhanced to detect the dangerous X-messages that mobile code sends out and filtering them out.

3.3.2 The Advantages of the Architecture

Not only does the architecture offer reliable security through several independent and small reference monitors instead of a big and complex one, but it also enhance the system security through reducing the inconsistency in enforcing security policies. If different users define their own security policy according to their own understanding and enforce these security policies separately, the chance of the inconsistent security policies will be very big and the system will be much easier to compromise. In our architecture, the security policy is defined by the system administrator who has better knowledge of security than an ordinary end-user has. These security policies are also enforced by the centralized reference monitors. Therefore, the risk from inconsistent security is small.

The architecture further enhance the system security by reducing the risk of so-called social engineering as the responsibility of maintaining security belongs to the system administrators. Because they are more aware of security implications of a particular application request than an ordinary end-user, they are less likely to hand out the critical security privileges. Social engineering attacks on the end-user would not gain unauthorized resource access. For example, in the above implementation, the mobile code executing in a browser started directly by an internal end-user would not gain access to the external network since TPO only accepts connections from the middle compartments.

The system's usability will also be enhanced as the result of freeing end-users from managing and maintaining system security themselves. They do not need to have extensive knowledge of network security to browse the Internet safely.

4 Conclusion

This paper demonstrates how the Mandatory Access Control model and the privilege management mechanism can be used to support sandbox method. By labelling the resources and process, MAC offers logical information separation that would otherwise only be achieved through expensive hardware separation. These security mechanisms also provide the necessary support to construct various tamperproof security reference monitors and force all privileged resource accesses to pass through these reference monitors.

The vaulted browser architecture described in this paper also demonstrates that MAC and privilege management is a suitable platform to support administrative security management. This ability allows us to shift the responsibility of defining and maintaining system's security from the end-users to the system security administrator. Therefore, the risk of social engineering attacks and inconsistent security policies can be greatly reduced. The system's usability can also be enhanced.

The vaulted browser architecture described in the paper also introduces several specialised security reference monitors. Not only does this method let us construct small and reliable reference monitors by reducing the dependencies between various

security "prongs", but it also offers the flexibility to support various security policy models.

In addition, because the concept of MAC and privilege management is independent of operating system functionality, it can be integrated into various operating system platforms. This also enables the method and the vaulted browser architecture introduced in this paper to be deployed on many operating system platforms.

---

1 Marvin Schaefer, Sylban Pinsky etc. "Ensuring Assurance in Mobile Computing", Panel discussion, proceedings of 1997 IEEE Symposium on Security and privacy", May 1997, Oakland

2 Gary Mcgraw and Edward W. Felten, Java Security: Hostile applets, Holes, and antidotes, Wiley Computer Publishing

3 Robert Wahbe, Steven Lucco, Thomas E. Anderson and Susan L. Graham, " Efficient Software-based fault isolation", Proceedings of the Sym. On Operating System Principles, 1993

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

5 TCSEC, Department of Defence, "Trusted Computer System Evaluation Criteria", DoD 5200.29-STD, Dec., 1985

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

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

8 M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas and L. Jones, "Socks Protocol Version 5", RFC 1928, March 1996

9 Nigel Edwards, Owen Rees, "High Security Web Servers and Gateways", Proceedings of Six International WWW Conference, April 1997, Santa Clara, California