Computing and Informatics, Vol. 27, 2008, 931–951

DESIGN AND IMPLEMENTATION OF SECURITY OS: A CASE STUDY

Seong-Ki Kim, Hae-Sung Son, Sang-Yong Han

School of Computer Science and Engineering
Seoul National University
56-1 Shinlim, Kwanak
Seoul, Korea 151-742
e-mail: ditoman@chollian.net, imjoy@kdb.co.kr, syhan@pplab.snu.ac.kr

Manuscript received 7 September 2007; revised 23 October 2007
Communicated by Prabhat Kumar Mahanti

Abstract. The importance of a security operating system (OS) with security-enhancing mechanisms at the kernel level, such as a reference monitor and a cryptographic file system, has been increasingly emphasized as the weaknesses and limitation of mechanisms at the user level have been revealed. However, when a system has only a reference monitor, the system is vulnerable to a low-level detour or a physical attack. In addition, when a system has only a cryptographic file system, the file system has a difficulty in protecting itself. To address these problems, we designed and developed a security OS with a reference monitor, a cryptographic file system, authentication limitation, and session limitation. Here we describe the model, its implementation, and its overheads.

Keywords: Security OS, security kernel, access control, cryptographic file system, PAM

1 INTRODUCTION

Many security breaches have occurred since the development of computer systems. In addition, as more systems have recently been connected, they have been exposed to a greater number of dangers. To avoid these threats, technologies such as firewalls, intrusion detection systems (IDS), and audit trails have flourished. However, these technologies have their own limitation. For instance, a firewall cannot protect

a system against an internal intruder. An audit trail cannot prevent an intrusion, but can only help subsequently trace an intruder by leaving messages. Commonly, all of these user-level mechanisms cannot protect themselves from being killed by attackers who already gained an administrator privilege because they all operate at the user level. In summary, the mechanisms only at the user level are too insufficient to protect a system and its important data.

These problems lead many researchers and vendors to place a focus on security OS with underlying mechanisms, such as a reference monitor, and a file system. The security kernel is one of the main security OS elements and includes a reference monitor, sometimes termed as an access control, which determines which subjects can perform which operations to which objects. However, the protection through reference monitors only has the weaknesses that they have additional overheads and are vulnerable to both low-level and physical attacks. As an example of overheads, one of access control implementations requires more than 5 % overheads that are intolerable in some cases (to be discussed in more detail in Subsection 3.4). As an example of both low-level and physical attacks, if an intruder finds a way to access information below the level of a reference monitor, then the intruder can access all information. In addition, if an intruder can detach a disk from a protected system, physically attach it to another system without a reference monitor, and mount it, then all information can be revealed.

The protection through a cryptographic file system only has also limitation when protecting a system and important data. For example, when a system administrator privilege is taken by an intruder in the system with the cryptographic file system only, the intruder will have more chances to read important data through finding the information on the installed cryptographic file system such as mounted directories, encryption algorithms, and key information because the system administrator can access all objects in the system. In other words, the protection through a cryptographic file system only is not enough without strong protection on system privileges.

Beside these problems, the aforementioned underlying mechanisms cannot prevent the various network attacks. When a user privilege is taken by an intruder at the system that is protected only by username and password, the intruder can always connect to the system at any hidden locations and try to do harms to the system. In addition, when an intruder tries to open many sessions, denial-of-service is caused and a normal user cannot connect to the system due to excessive overheads.

Many aforementioned problems can be solved by using them together. The low-level and physical problems of the reference monitor can be solved by using together the cryptographic file system. In this case, even if an intruder accesses information below the level of an access control or steals the protected disk, the intruder cannot view the information because of encryption. This weak privilege problem of the cryptographic file system can be solved by separating the security administrator from the system administrator, and giving cryptographic file system privileges to only the security administrator through a reference monitor. In this case, the reference monitor minimizes the intruder's chances to find the information on the

cryptographic file system even if the intruder gains a system administrator privilege, and effectively protects the important data in cooperation with cryptographic file systems. The network access problems of the underlying mechanisms can be solved by authentication limitation and session limitation. By the authentication limitation allowing the specific user to use the protected system at a specific place only, the system is protected better because any intruders cannot connect to the system if the intruders are not at the place. By the session limitation allowing only the limited number of connection sessions, an intruder cannot open more sessions to the protected system than the permitted number, which disables the denial-of-service attack. The authentication limitation and the session limitation are also protected because the reference monitor hides their information. In summary, when a reference monitor is used with a cryptographic file system, authentication limitation and session limitation, these mechanisms complement each other while making the system more invulnerable to attacks.

We designed a security OS with a new, simple, mandatory, fixed, and fast reference monitor, a cryptographic file system, authentication limitation, and session limitation. This paper describes the design, the implementation, and the overheads of our developed security OS for helping and contributing the other development of the security OS.

This paper is organized as follows. Section 2 refers to various related works by other researchers. Section 3 describes our security OS, Section 4 describes the example security policy configurations, and Section 5 reports its performance. Section 6 concludes.

2 RELATED WORKS

This section describes PAM [1], Cryptfs [2], RBAC [3, 4, 5], and SELinux [6].

2.1 Pluggable Authentication Module (PAM)

PAM was an application-independent authentication mechanism that was developed by Vipin Samar and Charlie Lai, and provided a system with the ability to support various authentication, account, session, and password pluggability features. Login applications using the PAM operate as shown in Figure 1.

Whenever applications such as ftpd, telnetd, and login require an authentication, the applications invoke PAM API, which loads appropriate authentication modules determined by the configuration files. After loading the modules, the authentication request is forwarded to the loaded modules, which returns the authentication legality to the requesting application [1].

2.2 Cryptfs

Cryptfs is a stackable cryptographic file system that is a part of the FiST toolkit [7] and exploits vnode stacking [8] to operate over various file systems. Cryptfs supports

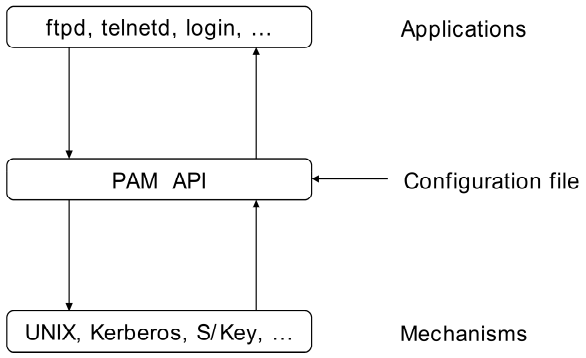

Fig. 1. PAM architecture

only one cipher, Blowfish [9], and implements a limited key management scheme. Figure 2 shows the call path of Cryptfs.

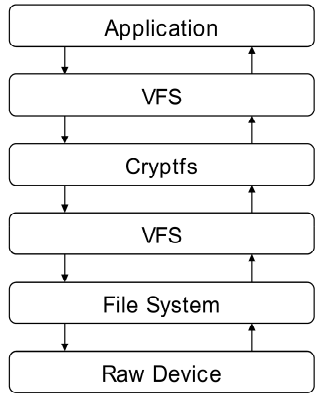

Fig. 2. Call path of Cryptfs [10]

Whenever an application or a user makes a request to the file system, the request is delivered to a vnode level call, which invokes its Cryptfs equivalent. Cryptfs internally invokes the original file system call through the vnode level calls, and encrypts or decrypts the contents.

Although Cryptfs is fast and secure because it is implemented at the kernel level, the protection only through the Cryptfs has the weakness that an intruder who gains system administrator privilege has more chances to access the protected information because the system administrator can access all information such as mounted directories, and key information. In addition, the intruder can stop the encryption because the Cryptfs has no mechanisms to control the system calls and cannot protect itself [2]. In other words, the protection through Cryptfs only is insufficient to powerfully protect a system.

2.3 Role Based Access Control (RBAC)

RBAC was the basic model to be referenced in order to design a reference monitor in this paper. RBAC supports mandatory control, and dense privilege. Basic RBAC has the model as shown in Figure 3.

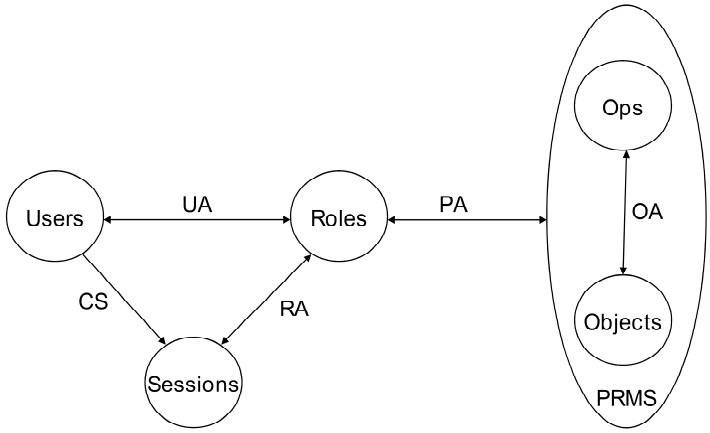

Fig. 3. Basic RBAC model [4],[11]

RBAC has elements such as users, roles, objects, sessions, and operations. A user represents the person who uses the system. A role has sets of both operations and objects determined by a security administrator. The role is given to users. A session represents the connection to a system with some of the roles activated during the session.

The cardinality between users and roles is many-to-many. In other words, a user can have many roles, and a role can be assigned to many users. The same principle is also applied to the cardinality between roles and sessions, as well as between operations and objects. A session can have many activated roles, and inversely a role can also be activated during many sessions. More than one operation can be applied to more than one object. The permutation of operations and objects can be thought of as a parameter assigned to roles. A role can have many parameters. A parameter can be assigned to many roles. However, the cardinality between users and sessions is one-to-many. In other words, although a user can have many sessions, a session can be involved with only one user.

A RBAC supports role inheritance to relieve the burden of the security administrator. Role inheritance supports a role to be created by inheriting another role. In this case, the derived role has the same sets of subjects' operations to objects as those of the base role. A RBAC also supports role constraints to place a restriction on the activated roles. As examples of role constraints, a mutually exclusive role constraint guarantees only one of the roles to be activated, a prerequisite role constraint guarantees prerequisite roles to be activated, and a cardinality role constraint guarantees the limited number of roles to be activated [3, 4, 5, 11].

2.4 Flask Architecture and SELinux

The National Security Agency (NSA) and Secure Computing Corporation (SCC) developed SELinux (security-enhanced Linux) [6] that implemented a Flask architecture [12], the goal of which was to support both flexible and transparent architecture for a variety of access controls. This goal was achieved by separating a security server from an object manager as shown in Figure 4.

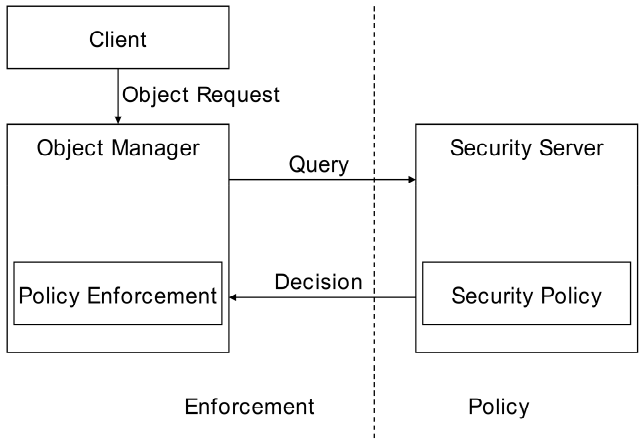

Fig. 4. The Flask architecture [12]

If a client wants to access an object, the client first makes a request to the object manager. The object manager determines whether the access is legal or not after querying to the security server that determines the legality based on security policies such as identity based access control (IBAC), RBAC, and type enforcement (TE). Because the security server can be replaced by another server with a different policy, SELinux can support all security policies as far as the security server is implemented while following the predetermined interfaces.

Although the Flask architecture was referenced during the early stage of a reference monitor, the Flask architecture was excluded because it had additional overheads related with communication cost between an object manager and a security server (this overheads issues will be discussed more detail in Subsection 3.4).

3 ARCHITECTURE AND IMPLEMENTATION

This section describes an architecture, and an implementation of a Security OS.

3.1 Overall Architecture

The overall architecture is shown in Figure 5.

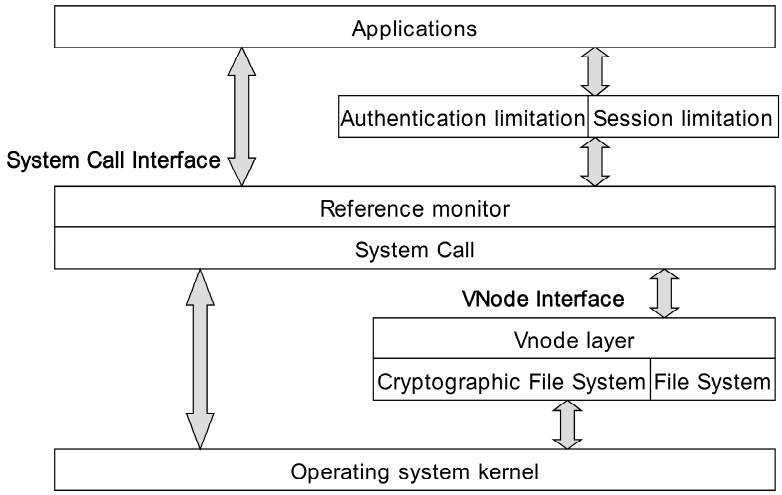

Fig. 5. Overall architecture

Whenever a user logs into a system, the login applications call authentication and session limitation to check a user identity, a password, an access location, and a session count. If the checks are successful and system access is granted, the object access passes through the developed reference monitor whenever the user accesses an object through an application. After the reference monitor checks whether an access is legal or not, it calls the original system call only if legal, which calls the developed cryptographic file system that encrypts or decrypts its contents only if the system call is either read or write file operations. Before decrypting and after encrypting, the cryptographic file system calls the Vnode layer that invokes the functions in the original file systems such as UNIX file system (UFS) and Network file system (NFS).

Only the reference monitor was firstly considered as a security OS. However, the reference monitor could not protect a system from a low-level detour or a physical theft, and the cryptographic file system was integrated in order to solve this problem. Also, the reference monitor could not protect a system from network attacks, and both the authentication limitation and the session limitation were added in order to solve this problem.

As the data protection examples of this architecture, we can apply a cryptographic file system to the important directory that needs to be protected, and give object-accessing operations to the users only to the extent that the users require for their own roles through the reference monitor. When these policies are applied, the reference monitor allows only specific users to read the important directory, and an intruder has difficulties in finding read-permitted users to the directory, gaining identities of the permitted users, and accessing the information on the encrypted directory. Even if the intruder can steal the disk and attach it to another system, or detour the reference monitor, the intruder has a difficulty in reading the important

directory due to the encryption through the cryptographic file system. As the network protection examples of this architecture, authentication limitation and session limitation allow a user to log into a system in only the case that the user is at the specific location and the connected number of the user is less than the specific number. A user cannot log into the system if the user accesses at the not-granted locations or if the specified number of the user are already connected, which makes attacks more difficult. Although IP spoofing enables an intruder to deceive his IP address, the intruder has no method to know the permitted IP addresses because the directory with the permitted addresses is protected by the reference monitor. Denial-of-service attack is also disabled because the session limitation makes it difficult to open more sessions than the permitted count.

The mutual protection is as follows. The reference monitor protects the cryptographic file system by disabling an intruder to access the information on the file system. The reference monitor also protects the authentication limitation and the session limitation from being controlled by unauthorized users through hiding their information. The cryptographic file system protects an object from being accessed through a low-level detour or physical theft that can avoid the reference monitor. Authentication limitation disables an intruder to connect the protected system at the places besides the permitted places. Session limitation disables an intruder to open many sessions in order to cause a denial-of-service. In summary, the reference monitor, the cryptographic file system, the authentication limitation, and the session limitation complement each other to protect a system and precious resources.

We implemented this architecture using a Sun Ultra 10 device with an UltraSparc-IIi 440 MHz CPU with 512 KB cache, 256 MB RAM, 9 GB IDE hard drive, and running the Solaris 2.8 operating system.

Two architectures could be chosen to implement this system. The first method was to create a kernel containing new functions. The second method was to add new functions into a kernel using a dynamically loadable kernel module (DLKM). Most modern operating systems supported a DLKM. In addition, the first method was very difficult because most operating systems were not license-free. Therefore, we chose the second method – the dynamic implementation method. Besides kernel modules, we developed a server daemon and an administration tool for easy management.

3.2 Authentication and Session Limitation

We used PAM [1] to realize the authentication limitation and session limitation regardless of applications. The architecture of authentication limitation and session limitation is shown in Figure 6.

Login applications dynamically load the developed PAM module by the modified configuration file. After loading the module that includes both authentication limitation and session limitation, the login application invokes the functions in the module, which checks a user identity, a password, the client IP address, and the current session count. If the password is incorrect, the IP address is not permitted;

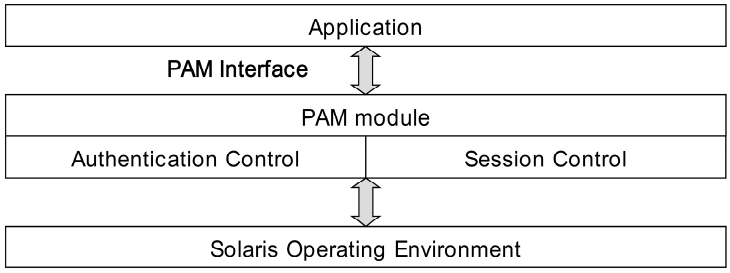

Fig. 6. The architecture of authentication limitation and session limitation

or if the session count exceeds the permitted number, then the login is denied. If the password is incorrect, the IP address is not permitted; or if the session count exceeds the permitted number, the login is denied.

PAM architecture was chosen because it enabled application-independent authentication. If PAM architecture had not been used, the source codes of the supported applications should have been searched, and modified to implement authentication and session limitation. Thus, the use of the PAM architecture made us avoid unnecessary and repetitive works.

The intruder has a hard time in attacking the system because the intruder doesnot know the permitted IP addresses, and finally cannot find the locations with the addresses. In addition, even if the system administrator privilege is gained by an intruder, the intruder cannot change the developed PAM module and skip the authentication limitation and session limitation because even the system administrator has no privileges to the PAM module.

3.3 Cryptographic File System

We used vnode stacking [8] at the kernel level to enhance the speed and the security. The architecture of the cryptographic file system is shown in Figure 7.

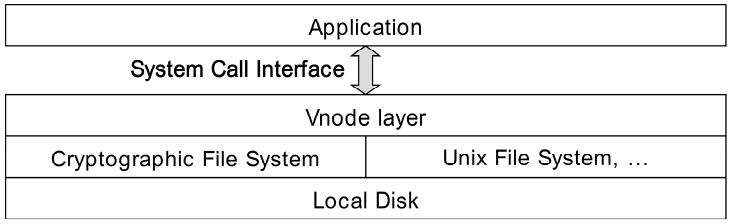

Fig. 7. Architecture of the cryptographic file system

An application or a user accesses a file through a read() system call, which calls a vnode layer through vn_read(). This vn_read() invokes the read() entry point in the developed cryptographic file system that reads from the other file systems and

decrypts the contents. The cryptographic file system has entry points such as read(), getpage(), write(), and putpage() that operate in this way.

The vnode stacking architecture at the kernel level was chosen because of low overheads without context switches, support of various file systems, and relative security of the kernel memory when compared with the user memory. The encryption units can be only the directories because the mounting points can be only the directories at the vnode stacking architecture.

As file encryption algorithms for objects, 128 bit Blowfish [9] and Seed [13] symmetric key algorithms were supported because of their speed, compact size, and Korean standard when compared with DES [9]. We felt that 128-bit keys were secure enough not to be broken for several years when considering the current CPU capacity. The key for the encrypted objects was delivered to the server daemon in Subsection 3.5 after randomly generated by the administration tool in Subsection 3.5 when a user wanted to use a cryptographic file system. The server daemon received the key, delivered it to the cryptographic file system, and saved it within a file after encrypting the received key. Whenever the server daemon was restarted after these initial processes, it delivered the key to the cryptographic file system after reading the key from the file and decrypting it without any necessities of receiving the key from the administration tool again. The encrypted key could be saved in a file on the disk with the developed security OS. In this case, the encrypted key was protected by both encryption and the policies of the reference monitor that allowed only security administrator to read the encrypted key. The encrypted key could be saved on a different removable disk from the protected disk with the security OS. In this case, the key was protected by both encryption and the security administrator.

This architecture is the same as that of Cryptfs in that this architecture uses the vnode stacking, and different from that of Cryptfs in that this architecture can save its keys in a file on the protected system or on the external storage system under the encrypted status, and supports a Seed algorithm in order to be used in Korea.

We newly implemented this architecture by adding the encryption and the decryption into lofs [14] because we could get its source codes, easily wanted to manage keys differently from Cryptfs, and wanted to support Seed algorithm for the usage in Korea. In addition, the lofs was supported by most of the UNIX.

3.4 Reference Monitor

Figure 8 shows the reference monitor model that was implemented to minimize the overheads.

This model has the elements such as subjects, permission, types, operations, and objects. A subject can be a user, a user group, or a process. A subject is similar to a user of the RBAC model in that the subject is the actor of object access, but different from the user of the RBAC model in that the subject can be a user group, and a process besides a user. Permission is the named set of types and operations. The permission is similar to the role of the RBAC model in that the permission has a name. An object can be a directory, or a file. This model has many-to-many

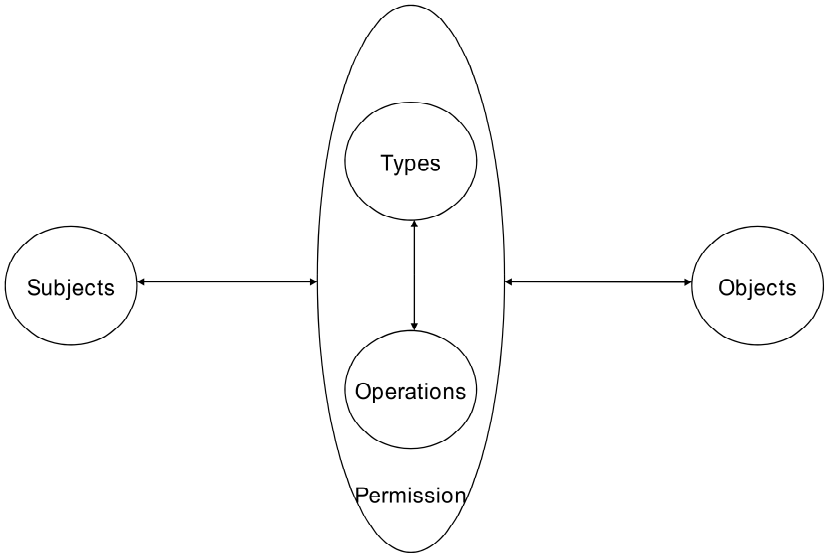

Fig. 8. Reference monitor model

relationships between permission and objects, and between permission and subjects. This model has many-to-many relationship between types and operations. In other words, permission can be assigned to many subjects, and a subject can have many permissions. Permission can be applied to many objects, and an object can have many permissions. As an example of this model, if permission has a read operation, it is applied to /a, and /b objects, and root and test users, then only root and test users can read /a, and /b files.

The standard RBAC model was firstly considered, but it was simplified and modified in order to minimize its implementation difficulties and its overheads caused by its complexity. A session, a role hierarchy, and role constraints were removed in this model also for simplicity and speed. Session and role constraints were not essential because we assumed that all users had the constant sets of permission during a long period. The assumption removed a session concept because a user did not have to have different roles based on sessions. The assumption also removed the concept of role constraints because all permission is always activated during a long period. Without the concept of role hierarchy, permission cannot be created by inheriting the other permission. Besides this inconvenience of a permission creation, a security administrator may have to often modify permission because this model does not support the permission activation based on sessions. As results of the removal of a session, a role hierarchy, and role constraints, the model is simple to implement, and fast.

From the viewpoint of security, this model is mandatory because only the security administrator can set the policies. Even the object owner cannot change his

or her granted operations without the permission by a security administrator. This model can support the least privilege through multiple permission assignments to subjects, and objects although the number of permission may be more than that of roles in the standard RBAC model.

This model is similar to the DAC model in that a user has permission to objects, but different from the DAC model in that this model does not maintain the ACL, has named permission, and is mandatory.

Table 1 shows the types and operations supported by permission.

Types and operations		Description
Types	INHERITANCE	All of the descendant directories and files inherit the subjects and the operations of the directory with this permission type.
Types	ALL	All of the users have the operations to the applied objects.
Types	OWNER	Object owners have the operations to the applied objects.
Types	OWNER GROUP	Object owner groups have the operations to the applied objects.
Operations	FORK	The subjects can create a process.
Operations	EXEC	The subjects can execute the processes distinguished by the applied objects.
Operations	KILL	The subjects can terminate the processes distinguished by the applied objects.
Operations	SETUID	The objects can change a user ID.
Operations	CHMOD	The subjects can change the modes of the applied objects.
Operations	CHOWN	The subjects can change the owners of the applied objects.
Operations	READ	The subjects can read the applied objects.
Operations	WRITE	The subjects can write the applied objects.
Operations	LINK	The subjects can create hard-links to the applied objects.
Operations	UNLINK	The subjects can unlink or delete the applied objects.
Operations	RENAME	The subjects can change the names of the applied objects.
Operations	MKDIR	The subjects can create descendant directories in the applied directory.
Operations	RMDIR	The subjects can delete descendant directories in the applied directory.
Operations	CHDIR	The subjects can change a current directory to the applied directory.
Operations	MOUNT	The subjects can mount file systems distinguished by the applied objects.
Operations	UNMOUNT	The subjects can unmount file systems distinguished by the applied objects.
Operations	MODLOAD	The subjects can load kernel modules distinguished by the applied objects.
Operations	MODUNLOAD	The subjects can unload kernel modules distinguished by the applied objects.
Operations	ROLE	The subjects can change a security policy.

Table 1. Types and operations of the reference monitor

Types are the inheritance or the subject properties of the permission that is created, and operations are checked actions. There exist types such as INHERITANCE, ALL, OWNER, and OWNER GROUP. INHERITANCE type means that the subjects and operations to the applied objects are inherited by all of the descendent objects. Even if new permission is applied to the descendant directory, the INHERITANCE type permission affects together with the new permission. No INHERITANCE type permission means that the permission affects the descendants

as far as the other permission is not applied. If an ascendant directory has permission without INHERITANCE and a descendant directory has new permission, the descendant directory is affected only by the new permission. If two or more permissions conflict each other, the OR is applied. In other words, if permission includes a read operation and another permission does not include a read operation, the read operation is granted by OR. ALL, OWNER, and OWNER GROUP types represent that all users, owner, and owner group, respectively are the subjects. All of the operations mean that subjects can do the operations to the applied objects as shown in Table 1.

Most of the operations such as CHMOD, CHOWN, READ, WRITE, LINK, UNLINK, RENAME, MKDIR, RMDIR, CHDIR, MOUNT, and UNMOUNT in Table 1 could be checked also by the cryptographic file system in Subsection 3.3, but the reference monitor checked the operations because the cryptographic file system was operated only when mounted, but the reference monitor was always operated during all accesses and all accesses should be checked.

The implementation of the reference monitor model is logically divided into five modules: a system call control module, an authentication module, a permission management module, an audit module, and an access control module as shown in Figure 9.

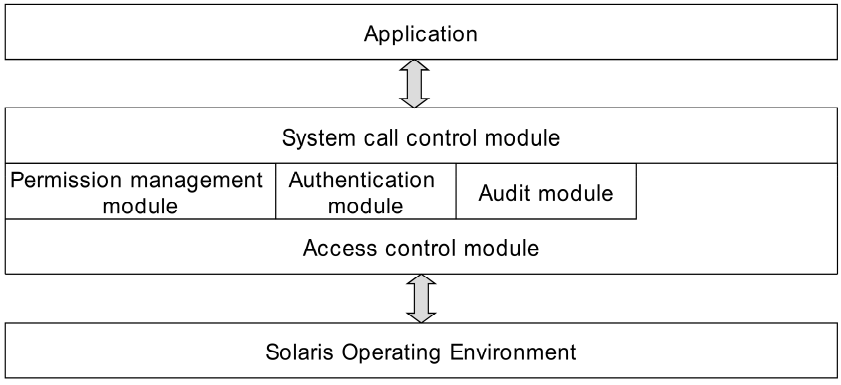

Fig. 9. Architecture of the reference monitor

The system call control module stores, hooks, and restores old system calls. In addition, it provides new system calls that assemble all of the other modules. The permission management module adds and deletes permission, modifies the types and the operations of permission, applies objects to permission, and assigns permission to subjects. The authentication module manages authentication information that is used for the access control module and is sent by the developed PAM module. The audit module records all accesses of users. The access control module determines the legality of an access. These five modules are logically separated just for understandability, and are included in one file.

In contrast to the double call architecture of SELinux in Section 2.4, we adopted this direct call architecture in order to minimize the overheads caused by flexibility. The implementation consisted of one file without the separated security server, and had the fixed model. Most of RBAC implementations concentrated not on compactness and speed, but on flexibility to support various policies. SELinux was a representative example. In SELinux, whenever a client wanted to access an object, the client made a request to an object manager, which in turn made a request to the security server to determine the legality of the access. As a result, the overheads of SELinux [6] were more than about 5 % in the Unixbench [15] case [6], about 10 % in the Lmbench [16] case [6], and about 4 % in the kernel compilation case [6]. Although an object manager maintained the AVC [12] to minimize these overheads in the double call architecture, AVC was insufficient to minimize all of these overheads.

3.5 Server Daemon and Administration Tool

For easy administration, we wanted to develop an administration tool on Windows. However, because the cryptographic file system and the reference monitor were kernel modules that had a difficulty in directly communicating with an administration tool, and both authentication limitation and session limitation were libraries, an additional server daemon needed to be developed in order to communicate with an administration tool. The server daemon received a request from the administration tool and then made a request to the authentication limitation, session limitation, cryptographic file system, and reference monitor through system calls. Another role of the server daemon is to periodically call the audit module via a new system call to obtain success or failure logs in the kernel memory, and to write them to files.

An administration tool was developed with Visual C++ version 6.0 on Windows XP. The administration tool received inputs from a user. The inputs were to mount a directory with Blowfish or Seed, unmount a directory, create permission, change either its types or operations, apply it to objects, assign it to a user, a user group or a process, change an access location, or change a session count. In all of these cases, the administration tool sent messages to the server daemon, which in turn made a request to the cryptographic file system, the reference monitor, the authentication limitation, or the session limitation.

The administration tool used an encrypted channel to communicate with the server daemon. The administration tool was connected with a server daemon after a handshake that certificated each other, and exchanged a communication key. The handshake occurred again at intervals of 24 hours and a new key was used for the later communication. In order to realize these processes, we used OpenSSL library [18].

3.6 Module Protection

Since we implemented all modules as a DLKM, a library, and an application to avoid kernel changes, the modules could be unloaded. However, the operation to unload our modules should be prohibited to unauthorized users. Thus, modules protected themselves by hooking an unload system call and creating an unload permission, P_ROOT in Section 4. Only subjects included in the permission could unload the developed modules. The server daemon itself should also be protected from being killed by unauthorized users. Thus, the server daemon also protected itself by hooking the kill system call and using the application-killing operation to the daemon object.

3.7 Real-Time Monitoring

To immediately notify the security administrators of an intrusion, the developed security OS supports a real-time monitoring. The administration tool periodically sends a message to the managed server daemon every 10 seconds. The tool receives real-time object and system access failure logs, and shows them at the administration tool. If these require urgent notification, the tool can notify the system administrator of the situation with a SMS message or an alarming sound.

4 SECURITY POLICY CONFIGURATIONS

Authentication limitation basically controls dtaction, dtlogin, ftp, login, su, pandorad, and OTHER services. X window login location is controlled by both dtaction and dtlogin. ftp, login, and su are the services that their names say, and serve the clients only with the permitted IP addresses. A pandorad service is the daemon service that communicates with an administration tool that controls the developed security OS. OTHER services refer to the other PAM-controlled services. Authentication limitation places no restrictions on the dtaction, dtlogin, ftp, login, su, and OTHER services. So, these services can connect with any IP address clients as a default. However, a pandorad service is connected only with an administration tool having the security administrator IP address, which prevents the server daemon from being connected with the administration tool having different IP addresses and protects against attacks imitating an administration tool.

Session limitation basically sets 5 as the maximum session count for every user, which minimizes the chance of being hacked by internet worms or intruders through opening too many sessions. Basically, the cryptographic file system is applied to no directories. But, if stronger security is needed by objects, the cryptographic file system can be applied to the directory and protect the objects cooperatively with the reference monitor. Table 2 shows the created permission for the normally operating Solaris 2.8 system.

These basic policies were created through examining the failed logs after prohibiting all accesses. ALL permission allows any users to freely access objects. It

Permission	Description
ALL	The objects with this permission can be freely accessed by anybody.
NULL	The objects with this permission cannot be accessed by anybody.
EXEC	The objects with this permission cannot be modified, but can be executed by anybody.
WRITE	The objects with this permission can be modified by anybody.
PASSWD_RD	The /etc/passwd cannot be modified, but be read by anybody.
PASSWD_WR	Only the subjects with this permission can modify /etc/passwd.
P_SETUID	Only the objects with this permission can invoke the setuid system call.
P_AUTH	Only the objects with this permission can change a security policy through the added system call.
P_CONF	Only the subjects with this permission can access the log, and the configuration files of the developed security OS.
P_ROOT	Only the root user can execute and terminate the server daemon of the security OS.
P_ALL	The security administrator and the server daemon can access any objects.

Table 2. Basic security policies of a reference monitor

has the ALL permission type, and all operations described in Table 1 except for MODLOAD, MODUNLOAD, SETUID, and ROLE. It is applied to both / and /usr/dt/appconfig directories. By this permission, all users can do the permitted operations to all of the descendant objects as far as the descendant objects have no other policies because the permission is applied to the / directory and is not the INHERITANCE type. If the other permission is applied to the descendant directory, ALL permission does not have any influence on the directory. The other permission is described in Table 2, and the subjects, types, operations, and objects are set in this way.

5 PERFORMANCE

Additional operation-checking functions were added by the new system calls in the reference monitor, encryption and decryption functions were added by the vnode operations in the cryptographic file system, and location- and session-checking functions were added by the authentication limitation and session limitation. Thus, we had no choice but to increase the overheads at the cost of enhancing a security.

This section describes the additional overheads using the benchmarks, Unixbench [15], and Lmbench [16] on a Sun Ultra 10 device with an UltraSparc-IIi 440 MHz CPU with 512 KB cache, 256 MB RAM, 9 GB IDE hard drive, and running the Solaris 2.8 operating system.

5.1 Unixbench

In Unixbench, system call performs a fixed number of system calls. Pipe throughput measures the communicating ability through a pipe between processes. Pipe-based context switching shows the communicating ability between a parent and a child. Process creation counts the number that a process can fork a child that exits. Execl throughput replaces a process with a new one. File copy measures the number of characters copied to a file within a given time. Lastly, shell scripts execute a shell script via concurrently running 8 processes [6, 15].

Figure 10 shows the Unixbench overheads when the system has only the reference monitor, both the reference monitor and SEED cryptographic file system, and both the reference monitor and Blowfish cryptographic file system. All of the numbers above the bars represent the overheads by percent (%) when compared with the general system without any developed modules. These overheads were measured after applying permission of the reference monitor and encryption of a cryptographic file system to a directory.

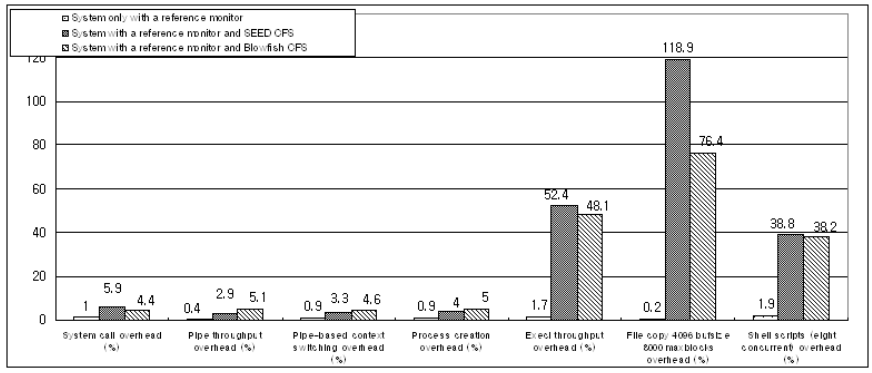

Fig. 10. Additional overheads measured by Unixbench when compared with a general system

In the system only with the reference monitor, the 1 % average overheads were related with the increased system call overheads by operation check whenever system calls were invoked. Pipe throughput and Pipe-based context switching invoked many read/write system calls. Process creation invoked system calls that created processes. Execl throughput and Shell scripts created additional authentication information, and invoked both read and process execution system calls during the measurement. The file copy overheads were low because the buffer size was huge enough to minimize the system call overheads.

In the system with both the reference monitor and file operation without decryption and encryption, the 4.4 % average overheads were caused by one more vnode layer. System call, Pipe throughput, Pipe-based context switching, and Process creation invoked read/write system calls. The average overheads of both the

reference monitor and file operations with decryption and encryption were 62.1 %. File copy had the most notable overheads among the file operations such as Execl throughput, File copy, and Shell scripts. During copying files, the reference monitor should check whether the user had read operation, the original file system should read its contents, the developed cryptographic file system should decrypt them, and the reference monitor should check whether the user had write operations, the cryptographic file system should encrypt them, and the original file system should write the encrypted contents. The other remarkable cases were Execl throughput, and Shell scripts. These cases were also related with the decryption overheads when reading files. Comparison of the overheads of Seed and Blowfish showed that the difference between them was small.

5.2 Lmbench

In Lmbench, simple syscall reads a byte from /dev/zero and writes it to /dev/null. Simple read and write perform 4 bytes read and write operations. Simple stat obtains file information without an open operation. Process fork+exit, Process fork+execve, and Process fork+/bin/sh -c perform the operations written and measure the time [16].

The Lmbench overheads are shown in Figure 11. All of the numbers above the bars also represent the overheads by percent (%). These overheads were also measured after applying permission of the reference monitor and encryption of the cryptographic file system to a directory.

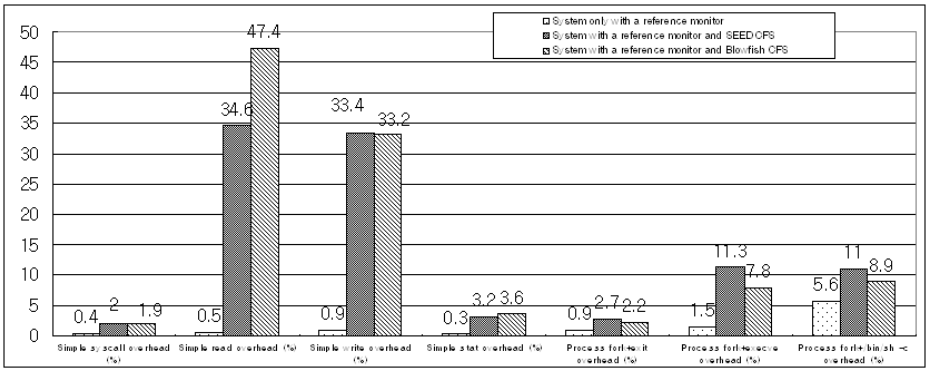

Fig. 11. Additional overheads measured by Lmbench when compared with a general system

In the system only with the reference monitor, the 1.4 % average overheads were related with the increased system call overheads by operation check whenever system calls were invoked. The overheads of the process fork+exit, fork+execve, and fork+/bin/sh -c cases were notable. These cases were affected by the system

call overheads of the reference monitor that created authentication information, executed a program, and searched the path.

In the system with the reference monitor and file operation without decryption and encryption, Simple syscall, Simple stat, and Process fork+exit have only the 2.6 % average overheads due to one more vnode layer. Simple read, Simple write, Process fork+execve, and Process fork+/bin/sh -c with decryption or encryption have 23.5 % average overheads. Comparison of the overheads of Seed and Blowfish showed that the difference between them was also small.

6 CONCLUSIONS

We designed a security OS with a reference monitor, a cryptographic file system, authentication limitation, and session limitation. The reference monitor was newly designed and implemented so that it could be fast, simple, mandatory, and fixed because the overheads of supporting all of the RBAC features and flexibility were significant. It had RBAC elements, DAC characteristics, and no need to change OS. The cryptographic file system, authentication limitation, and session limitation were added in order to complement each other and their overheads were measured. The average overheads were 1.2 % for the case only with the reference monitor, the average overheads were 3.5 % for file operation cases without encryption and decryption, and the average overheads were 42.8 % for file operation cases with encryption and decryption due to the operation-checking of the reference monitor, and the encryption and the decryption of the cryptographic file system. In most of the cases, the overhead was only 1.2 % because the file operations with encryption and decryption were applied to only the very important directories.

We concluded that the advantages compensate for the slight overheads. Our security OS gives better system protection: i) by the least privilege through the reference monitor; ii) by making it difficult to steal and access the protected disk through the cryptographic file system; iii) by making it difficult for an intruder to access the system in the not-permitted places; iv) by making it difficult for an intruder to attack a system through opening many sessions; and v) by making it difficult to unload or kill the developed modules or applications. As a research project, our model and implementation of a security OS are valuable because, to the best of our knowledge, they represent the first security OS that has the authentication limitation, the session limitation, the cryptographic file system, and the reference monitor.

REFERENCES

[1] Samar, V.—Lai, C.: Making Login Services Independent of Authentication Technologies. Proceedings of the SunSoft Developer's Conference, March 1996.

[2] Zadok, E.—Badulescu, I.—Shender, A.: Cryptfs: A Stackable Vnode Level Encryption File System. Technical Report CUCS-021-98, Computer Science Department, Columbia University, 1998.

[3] Hitchens, M.—Varadharajan, V.: Design and Specification of Role Based Access Control Policies. IEE Proceedings – Software, Vol. 147, August 2000, No. 4, pp. 117–129.

[4] Ferraiolo, D. F.—Sandhu, R.—Gavrila, S.—Kuhn, D. R.—Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and System Security, Vol. 4, 2001, No. 3, pp. 224–274.

[5] Ferraiolo, D. F.—Cugini, J.—Kuhn, D. R.: Role-Based Access Control (RBAC): Features and Motivations. Proceedings of The 11th Annual Computer Security Applications Conference, New Orleans, USA, December 1995, pp. 241–248.

[6] Loscocco, P.—Smalley, S.: Integrating Flexible Support for Security Policies Into the Linux Operating System. Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01), June 2001, pp. 29–42.

[7] Zadok, E.—Nieh, J.: FiST: A Language for Stackable File Systems. Proceedings of USENIX Annual Conference, June 2000, pp. 55–70.

[8] Heidemann, J. S.—Popek, G. J.: File System Development With Stackable Layers. ACM Transactions on Computer Systems (TOCS), Vol. 12, 1994, No. 1, pp. 58–89.

[9] Schneier, B.: Applied Cryptography. 2nd ed. John Wiley & Sons, 1996.

[10] Wright, C. P.—Dave, J.—Zadok, E.: Cryptographic File System Performance: What You Don't Know Can Hurt You. Proceedings of the 2003 IEEE Security In Storage Workshop (SISW 2003), October 2003.

[11] Koch, M.—Mancini, L. V.—Parisi-Presicce, F.: A Graph-Based Formalism for RBAC. ACM Transactions on Information and System Security (TISSEC), Vol. 5, August 2002, No. 3, pp. 332–365.

[12] Spencer, R.—Smalley, S.—Loscocco, P.—Hibler, M.—Andersen, D.—Lepreau, J.: The Flask Security Architecture: System support for diverse security policies. Technical Report UUCS-98-014, University of Utah, August 1998.

[13] Telecommunication Technology Association: 128-bit Symmetric Block Cipher (SEED), TTAS.KO-12.0004, South Korea, September 1998.

[14] SMCC: Lofs-Loopback Virtual File System, Sun Microsystems Inc., SunOS 5.5.1 Reference Manual, Section 7, 20 March 1992.

[15] Niemi, D. C.: Unixbench 4.1.0. Availaible on: `http://www.tux.org/pub/tux/niemi/unixbench`.

[16] McVoy, L.—Staelin, C.: lmbench 2. Availaible on: `http://www.bitmover.com/lmbench`.

[17] Mauro, J.—McDougall, R.: Solaris Internals Core Kernel Architecture. PH PTR, 2001.

[18] OpenSSL: The Open Source toolkit for SSL/TLS. Availaible on: `http://www.openssl.org`.

Seong-Ki Kim received the B. Sc. degree from Konkuk University, Seoul, Korea in 2000 in Computer Science. He participated in an E-commerce project, in a PACS project and in a multimedia system project from 2000 to 2001. He joined the School of Computer Science & Engineering at Seoul National University since 2002, and participated in the researches on data broadcasting and a security OS. He is currently Ph. D. candidate at Seoul National University, and conducts researches on a multimedia system. His current research interests include the areas of a multimedia system, a security OS, a parallel processing and a distributed computing.

Sang-Yong Han received the B. Sc. and the M. Sc. degrees from Seoul National University, Seoul, Korea in 1972 and 1977 in Applied Mathematics and in Computer Science, respectively. He received the Ph. D. degree in University of Texas at Austin in 1983. He has been a faculty at the School of Computer Science & Engineering at Seoul National University since 1984, and currently teaches a computer architecture, a parallel processing and a programming language. His current research interests include the areas of a security OS, a programming language, a parallel processing and a distributed computing.

Hae-Sung Son received the B. Sc. degree in Computer Engineering from Kyungpook National University, Taegu, Korea, in 2003, and M. Sc. from Seoul National University, Seoul, Korea, in 2005. She had participated in the researches on a secure OS during 2003–2005. She joined IT department of the Korea Development Bank, and participated in an e-L/C project and an OTP system project. She has worked on electronic banking and front-end processing system administration at the Korea Development Bank.