ON THE SECURITY OF SELINUX WITH A SIMPLIFIED POLICY

Katsuya Sueyasu
Department of Electrical Engineering and Computer Science,
Faculty of Engineering, Kyushu University
6-10-1 Hakozaki, Higashi-ku,
Fukuoka 812-8581, Japan
email: sueyasu@itslab.csce.kyushu-u.ac.jp

Toshihiro Tabata and Kouichi Sakurai
Faculty of Information Science and
Electrical Engineering, Kyushu University
6-10-1 Hakozaki, Higashi-ku,
Fukuoka 812-8581, Japan
email: {tabata, sakurai}@csce.kyushu-u.ac.jp

ABSTRACT
Security-Enhanced Linux (SELinux) is a secure operating system. SELinux implements some features in order to perform strong access control. However, the configuration of SELinux access control becomes very complex. Such complexity may cause misconfiguration which can harm the strong access control. SELinux Policy Editor is a configuration tool for SELinux. It is developed in order to reduce the complexity and the risk of misconfiguration. As a part of its support of configuration, this tool simplifies the configuration of SELinux by integrating configuration items for complicated access control policy of SELinux. Although we can originally define and use macros which integrate permissions in SELinux access control policy, the integrated permissions of SELinux Policy Editor and the macros differ fundamentally in whether the use of them is mandatory or discretionary. In this paper, we examine effects of the simplification by SELinux Policy Editor on an example access control policy and evaluate the security of the access control based on the simplified policy about Apache, a web server software.

KEY WORDS
Security for Operating Systems, Access Control, Security-Enhanced Linux, SELinux Policy Editor

1 Introduction

Since computer networks are growing rapidly, networked computers today are at higher risk of attack through network. For example, a malicious person may hack web pages or release computer worms which perform undesirable activities on victim machines. Because an operating system is the basis of a system, we need operating system-level security in order to prevent damage from these attacks fundamentally.

Security-Enhanced Linux (SELinux) is a Linux-based operating system developed by National Security Agency (NSA) [1] [2]. SELinux supports mandatory access control (MAC), type enforcement (TE), and role based access control (RBAC). These access control models can minimize permissions granted to each subject, so that activities of an attacker who intruded into a SELinux system can be strongly restricted. However, because of being implemented these models, the configuration of SELinux access control becomes very complex [3]. Such complexity may cause misconfiguration which can harm the strong access control.

To reduce this complexity, a configuration tool called SELinux Policy Editor is developed by Hitachi Software [4]. As a part of its support of configuration, this tool simplifies the configuration of SELinux by integrating some configuration items used in the SELinux access control policy [5]. For example, there are seventeen permissions for file access in SELinux. On the other hand, SELinux Policy Editor integrates them into four permissions. For example, “write,” “append,” and “create” are permissions defined in SELinux but they are all integrated into a “write” permission by SELinux Policy Editor.

In this paper, we start by comparing this simplification with a simplification by macros. We can use macros originally in configuring a SELinux policy and they are frequently used in the SELinux example policy indeed. NSA provides the example policy for policy administrators as a sample of configuration of major applications. Although users can define their own macros, we consider macros defined in the example policy in comparing the two simplifications.

Then, we examine effects of the simplification by SELinux Policy Editor on a file access control policy and evaluate the security of file access control based on the policy. Our evaluation is performed on the example policy for Apache, because it is a popular web server program used around the world and most actual SELinux policies are thought to be based on the example policy. Our aim is to point out problems that SELinux Policy Editor can cause in actual (or nearly actual) cases.

As related works, Tresys Technology also has been developing tools for SELinux [6]. Tresys develops not only a configuring tool but tools for policy analyzing or user management. These tools help policy administrators understand the SELinux policy and perform their tasks (e.g., adding a new user). Further, Jaeger et al. presented an approach for analyzing the integrity protection in the

SELinux example policy [7].

The rest of this paper is organized as follows. In section 2, we explain the function of SELinux. In section 3, we explain the function of SELinux Policy Editor. In section 4, we examine effects of the simplification by SELinux Policy Editor. In section 5, we evaluate the security of file access control based on the policy. Then, we conclude in section 6.

2 Security-Enhanced Linux

This chapter explains briefly the function of SELinux and the mechanism in which it is implemented.

2.1 Function of SELinux

SELinux implements strong access control. The first characteristic of the control is mandatory access control (MAC). In discretionary access control (DAC), a superuser (like "root" in Linux OS) exists and all access control will be disregarded. However, in MAC, even a superuser will be set as the object of access control. Moreover, although the owner of a file can determine the access authority over the file freely in DAC, the decision of access authority can be made only by the user allowed to do so in MAC. By carrying out centralized management of access authority, we can provide consistent access control.

The second characteristic is Type Enforcement (TE). Since we can define access authority for each process in TE, it is possible to minimize access authority granted to a process.

The third characteristic is Role-Based Access Control (RBAC). In Linux OS, a superuser (root) holds all the authority of important processing on a system management. For this reason, when performing an important processing as a superuser, a mistake can lead to destruction of an important file which is unrelated to the processing. In RBAC, since superuser's authority can be distributed to two or more general users, the damage produced by mistakes can be suppressed to the minimum.

These characteristics restrict activities of an attacker by distributing authority to users and removing excessive authority from each process. If a process is granted only necessary authority, an attacker who takes control of the process cannot deal a big blow to the target system.

2.2 Mechanism of SELinux

The access control mechanism of SELinux is made of adding the access control mechanism which has the function explained above in the outside of the access control mechanism of Linux OS. That is, we can make an access only after obtaining access permissions from both of mechanisms. We explain the additional access control mechanism here.

As shown in Figure 1, the mechanism assigns subjects (e.g., processes) labels called "domain" and objects (e.g., files) labels called "type." Each subject (object) has one domain (type). The configuration of SELinux access control is based on these label names. For example, the example description in Figure 2 allows subjects with domain D to access files with type T in three ways (get attributes, read, or append). A label name is a character sequence ended by "_t." When the same label is assigned to two or more objects (subjects), they will have equivalent access authority information. Since we can assign different domains to each process, it turns out that TE is realizable.

"Role" is related with users. We can relate one role with two or more users, and we can also relate two or more roles with one user. A user chooses a role at the time of login, and user shell is started in a domain corresponding to the role. Since different roles can grant different authority to user shell, it turns out that RBAC is realizable.

Next, we explain domain transitions which are methods for assigning domains to processes. Except for an initialization process, all the processes are created as a child process by a parent process. If there is especially no specification, a child process will inherit the domain of a parent process. However, if a domain transition is defined clearly, a child process will be created with another domain. In this way, we can grant different authority than its parent's to the child process.

All of these configurations are performed by editing an access control configuration file. Since it means that only users with the authority for editing the configuration file can perform an access control configuration, it turns out that MAC is realizable. Moreover, any access authority must be clearly described in the configuration file. That is, any access without such a description will be denied.

Thus, fine access control over every process and user can be performed in SELinux. Furthermore, the kind of permission has also increased compared with Linux OS. However, a fine configuration takes a labor so much. In order to cut down the labor, we can use macros in SELinux. Although a user can define his own macros, they are defined also in the example policy.

3 SELinux Policy Editor

An access control configuration of SELinux is performed by editing the configuration file. Since it is hard to get hold of the present state of configuration from the text-based configuration file, an omission or an error may be in the configuration result. If such a defect is in a configuration,

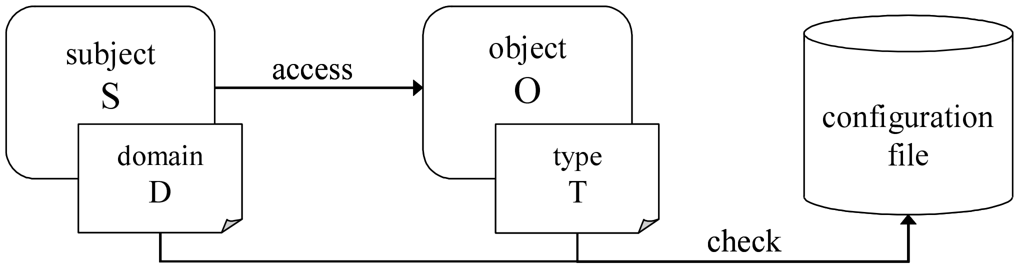

Figure 1. Additional access control mechanism in SELinux

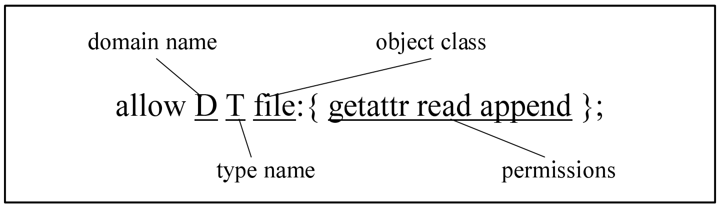

Figure 2. Example description of authority

strong access control of SELinux will be harmed. In order to reduce the harm, Hitachi Software is developing a configuration tool called SELinux Policy Editor.

We briefly explain the function of SELinux Policy Editor here. First, SELinux Policy Editor enables us to configure with GUI (Graphical User Interface). With GUI, we can visually get hold of the present state of configuration (e.g., domain transition). Second, SELinux Policy Editor uses an original configuration language. A user edits a configuration file in the language with GUI, then SELinux Policy Editor converts the file into the configuration file which is valid for current version of SELinux. Since the original language never depends on the version of SELinux, a user does not need to be conscious of the current version of SELinux. Third, some permissions are integrated. Although there are seventeen kinds of permissions for file access in SELinux, SELinux Policy Editor integrates twelve of them into four permissions which are frequently used. In this way, the configuration can be simplified.

4 Policy Simplification

In this section, we examine effects of the simplification by SELinux Policy Editor. As an example, we consider configuring a policy as similar as possible to the example policy for Apache by using SELinux Policy Editor.

4.1 Comparison of Simplification Methods

Both macros in SELinux and integrated permissions in SELinux Policy Editor simplify the configuration by reducing the kind of permission. First, we compare these two simplification methods.

These methods differ fundamentally in whether it is discretionary or mandatory. Although use of macros is discretionary, a user is forced to use the integrated permissions in SELinux Policy Editor.

Though, in the example policy, macros are frequently used very much. For example, 90% or more of “read” permissions about file access in the example policy comes from macros. Therefore, when trying to configure the example policy with SELinux Policy Editor, we often consider expressing macros by using integrated permissions. Table 1 and 2 show correspondences between macros and integrated permissions for file and directory access. Since normal and necessary access may be denied if authority runs short, the correspondences are expressed so that a combination of integrated permissions includes at least all permissions which come from a macro. In the tables, “r” means “read,” “w” means “write,” “a” means “append,” “x” means “execute,” and “s” means “search.” The macros named “create” include permissions which allow modification not only to contents of a file or a directory but to their information (e.g., name, last modified date and time, or even existence). Write-dir-perms macro includes permissions which allow modification to contents of a directory, that is, the macro enables us to make or delete a file or

Table 1: Correspondence between macros and integrated permissions for file access

macros	integrated permissions				excessive permissions
	r	w	x	s	
x_file_perms	×	×	○	○	-
r_file_perms	○	×	×	○	-
rx_file_perms	○	×	○	○	-
rw_file_perms	○	○	×	○	setattr, create, link, unlink, rename
ra_file_perms	○	○	×	○	setattr, create, link, unlink, rename, write
create_file_perms	○	○	×	○	-

Table 2: Correspondence between macros and integrated permissions for directory access

macros	integrated permissions				excessive permissions
	r	w	x	s	
r_dir_perms	○	×	×	○	-
rw_dir_perms	○	○	×	○	setattr, create, link, unlink, rename, reparent, rmdir
ra_dir_perms	○	○	×	○	setattr, create, link, unlink, rename, reparent, rmdir, remove_name
create_dir_perms	○	○	×	○	-

a directory directly under the directory.

Table 1 and 2 show that there is no combination of integrated permissions which exactly corresponds to macros named “write” or “append.” In other words, for example, write-file-perms macro is integrated into create-file-perms macro in SELinux Policy Editor. It means that a process which can edit a file can also modify the information of the file in any case. However, we can say that they correspond well about other macros.

4.2 Example of Simplification

As an example of policy simplification, we consider trying to configure the example policy for Apache with SELinux Policy Editor, and we examine the difference between the policy configured and the example policy. Although SELinux also controls operations on processes and so on by permission, in this paper, we consider only access control of files and directories.

Macros are frequently used in the example policy for Apache, too. We can make the integrated permissions correspond to the macros exactly, except those which are named “write” or “append.” Therefore, since the simplification by SELinux Policy Editor never affects permissions which come from such macros, we except them from examination.

When Apache starts, httpd process is created and httpd_t domain is assigned to the process by a domain transition. That is, even if an attacker exploits an unknown vulnerability of Apache and takes control of a process, the domains which the attacker can obtain will be limited to httpd_t and domains which httpd_t can make transitions to, as shown in Figure 3. Therefore, we consider the configuration only about these domains.

Based on the above, we manually analyzed the example policy. As a result, it turned out that there are two main kinds of effects that the simplification by SELinux Policy Editor causes.

One effect is related to macros named “write” or “append,” as described above. Examples of files and directories which may be affected by the effect are as follows.

- cache files and directories
- log files and directories
- temporary files and directories
- files and directories which CGI scripts access
- directories for CGI scripts

Another effect is related to “read” and “search” permissions for directory access. In SELinux, when a process accesses a file or a directory, the process must be granted not only permissions on the file or the directory but “search” permissions on all directories on the path. On the other hand, “read” permission for directory access is used to know what files or directories are directly under a directory. These two permissions are integrated into the integrated permission “s” in SELinux Policy Editor. However, in the example policy for Apache, some domains are granted only “search” permission on a type. Therefore, with SELinux Policy Editor, such domains will be granted excessive “read” permissions on the type. Examples of files and directories which may be affected by the effect are as follows.

- /boot directory
- directories for web pages
- directories for CGI scripts
- users’ home directories

5 Security Evaluation

In the previous section, we examined effects of the simplification by SELinux Policy Editor on the example policy for Apache. In this section, we examine whether these effects harm the security of file access control of SELinux. We assume that the attacker can obtain httpd_t domain and domains which httpd_t can make transitions to and execute any code in these domains.

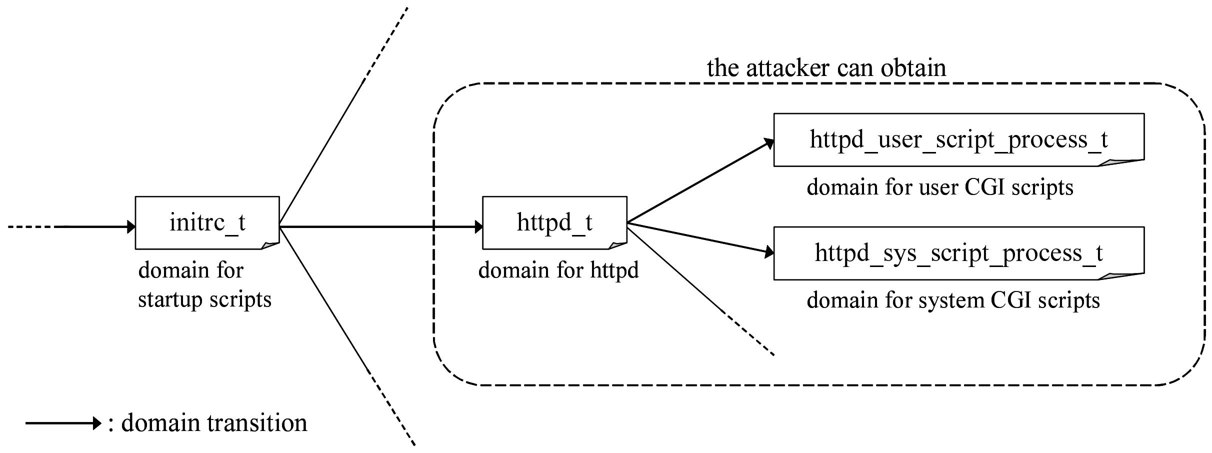
the attacker can obtain
initrc_t
domain for startup scripts
httpd_t
domain for httpd
httpd_user_script_process_t
domain for user CGI scripts
httpd_sys_script_process_t
domain for system CGI scripts
: domain transition

Figure 3. Domains which the attacker can obtain from Apache

If a domain is granted a macro named "create" on a type, we can disregard the effect of macros named "write" or "append" and granted to the other domains on the type. The attacker can make an access in a domain granted the stronger macro named "create" instead of "write" or "append." For example, a domain in which CGI scripts run is granted ra_file_perms macro on log files of Apache. However, since the httpd daemon must manage these log files, httpd_t domain is granted create_file_perms macro on them.

Similarly, r_dir_perms macro includes "read" and "search" permissions for directory access. Therefore, if a domain is granted r_dir_perms macro on a type, we can disregard the effect of "search" permission granted to the other domains on the type.

When we disregard such effects, the former list of files and directories in section 4 changes as follows.

- /var/cache directory
- log directories
- append-only files and directories which CGI scripts access
- directories for CGI scripts

The latter list changes as follows.

- /boot directory
- users' home directories

We examine these factors in turn.

If Apache is used as proxy, httpd directory is created under /var/cache directory as a cache directory. Moreover, since /var directory is assigned the same type as /var/cache directory in the example policy, domains are granted the same authority to /var directory as the authority to /var/cache directory. As the result of the policy simplification by SELinux Policy Editor, some domains become to be allowed to delete these directories. However, when we delete a directory, we must delete all the objects in the directory in advance. Since the attacker is not allowed to access most files under /var directory (especially files unrelated to Apache), what the attacker can newly do is to delete a directory which is already empty at the most. The same is said of log directories.

The attacker is originally allowed to append to append-only files and directories which CGI scripts access. Appending to a directory means only creating a file or a directory in the directory. As the result of the policy simplification by SELinux Policy Editor, the attacker is also allowed to delete these files and directories. The same is said of directories for CGI scripts. However, because CGI scripts themselves are originally read-and-execute-only, it is impossible for the attacker to delete directories for CGI scripts.

Files in /boot directory are used to start the OS. The configuration of this directory is performed in common among general domains. As the result of the policy simplification by SELinux Policy Editor, the attacker is allowed to obtain the list of files and directories in the directory. However, this does not mean that the attacker becomes to be able to modify or delete these files or directory. The same is said of users' home directory.

In SELinux Policy Editor, a type name depends on the path name of an object to which the type is assigned. Thus, SELinux Policy Editor always assigns a different type to each file and directory. Therefore, in the simplified policy, we can easily remove added permissions from Apache-unrelated objects independently. However, we cannot conceal Apache-unrelated objects directly under directories in the path of an Apache-related object because a "read" permission on such directory must be granted at the same time with a required "search" permission. Moreover, we cannot reduce the effect of the problem of deletion of append-only objects because the permissions for append-only access are completely integrated.

That is, possible effects of SELinux Policy Editor on the security of SELinux are as follows.

- risk of modification or deletion of some append-only files or directories which Apache accesses.
- risk of exposure of lists of some directories (e.g., users' home directories)

The risk of modification or deletion can be thought only about Apache-related and append-only objects. The security for the other objects is still maintained with SELinux Policy Editor. If there is an object whose existence must be concealed in a system, the integration of "read" and "search" permissions may harm the security of the system. However, we can minimize the risk by putting such a object in an appropriate place and by making an additional configuration.

6 Conclusion

In this paper, we examined effects of the simplification by SELinux Policy Editor on the example policy for Apache and evaluated the security of SELinux with the policy about file access control. Then, we explained how SELinux Policy Editor affects the security of SELinux.

As a future work, we will examine on other examples and consider effects of the simplification generally, so that we can study how we can integrate the permissions of SELinux in order to simplify the configuration while we never harm the security.

References

[1] Security-Enhanced Linux
URL=http://www.nsa.gov/selinux/

[2] P. Loscocco and S. Smalley, Integrating Flexible Support for Security Policies into the Linux Operating System, *Proc. of the FREENIX Track of the 2001 USENIX Annual Technical Conference*, Boston, USA, 2001, 29–42.

[3] S. Smalley, Configuring the SELinux Policy, NAI Labs Rep. 02-007, 2003.
URL=http://www.nsa.gov/selinux/policy2-abs.html

[4] SELinux Policy Editor
URL=http://www.selinux.hitachi-sk.co.jp/en/tool/selpe/selpe-top.html

[5] Y. Nakamura and Y. Sameshima, Configuration system for access control policy of Security-Enhanced Linux, *Proc. of the 2003 Symposium on Cryptgraphy and Information Security (SCIS 2003)*, Shizuoka, Japan, 2003, Vol. II, 831–836.

[6] Tresys Technology. SELinux research.
URL=http://www.tresys.com/selinux/index.html

[7] T. Jaeger, R. Sailer, X. Zhang, Analyzing Integrity Protection in the SELinux Example Policy, *Proc. of the 12th USENIX Security Symposium*, Washington D.C., USA, 2003, 59–74.