On the Privilege Transitional Attack in Secure Operating Systems

Hyung Chan Kim† R. S. Ramakrishna† Kouichi Sakurai‡

†Department of Information and Communications,
Gwangju Institute of Science and Technology (GIST),
1 Oryong-dong, Buk-gu, Gwangju 500-712, Rep. of Korea

{kimhc, rsr}@gist.ac.kr

‡Faculty of Computer Science and Communication Engineering
Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan

sakurai@csce.kyushu-u.ac.jp

Abstract In conventional Unix/Linux systems, getting the superuser permission exploiting setuid programs is prevalent threatening the given system. In this paper we examine the possibility of privilege transitional attack in secure operating systems hardened by logical access control and discuss on the prevention issues. We perform our experiment on general Linux and SELinux system.

1 Introduction

The improvement of exploiting techniques on Linux/Unix operating systems has been accelerated. After the publication of Buffer Overflow technique by Aleph One [1], many other variations are involved such as exploiting Static or BSS area, Double Free, Format String and so on. All these techniques can be acquired from the internet easily. Some of them directly affects to the practical daemon application accompanying the privilege transition of the attacker. Recently, as the analysis of Linux kernel code becomes easier – the code is opened and many resources are available in public – attack codes wreak havoc on the kernel directly and in this case the harm is more severe.

There have been many efforts to cope with these attacks. For example, works on the protection against buffer overflow are as follow: fetching vulnerable functions in the library level [2], coding a canary word in the near return address [3], preserving a return address in the text area and comparing with it when the function call is over [4].

Beyond the protection mechanism in the user process level, Secure Operating Systems (Trusted OS), which harden the security in the kernel level, are emerged in both of research and commercial field. Conventional Unix OS implements discretionary access control (DAC) wherein an owner of file can transfer the access right of a file to the others. Whereas, Secure OS offers level-based or role-based access control under the control of central security administrator and this is a type of mandatory access control (MAC). Upon this kind of logical access control, there are some efforts, such as GRSecuiry and Pax [5, 6], to support several architectural protection mechanism in kernel level.

If Linux OS contains a bug program with setuid assignment, an attacker can exploit it and run privileged shell with associated uid. Surely if the uid is zero, it means that the attacker can attain the whole right of the system in sense of DAC.

The similar attack is possible in Secure OS. This paper shows that two experimental results of program attack in generic Linux and SELinux which is hardened by RBAC and TE.

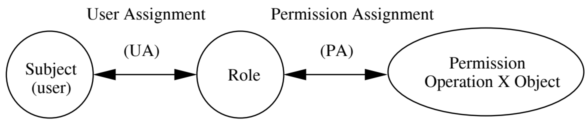

Figure 1: Role-Based Access Control

We make the condition in SELinux similar with setuid in conventional Linux so as to show the same program, which includes a runtime bug, can harm both generic Linux and Secure OS.

2 Background

2.1 Role-Based Access Control

Here we briefly introduce the concept of RBAC [8, 12], because it is widely adopted in Secure OS.

The main characteristic of RBAC is that it does not directly associate a subject with an object. Instead, by conceiving the *role* which represents job functions or responsibilities in a system or an organization [Fig. 1], RBAC greatly eases access control administration. A conventional DAC or a MAC system usually involves direct association between a subject and an object. If there are hundreds of thousands of access entities – a possibility in large enterprises – administrators of DAC or MAC system have difficulty in managing all the access entities. A specific role gathers a set of necessary permissions – defined as the cartesian product of the set of operations and the set of objects – in order to perform a certain duty. Hence, if an administrator of the RBAC system wants to make a subject perform a given duty, then the subject is simply assigned an appropriate role.

The abstraction of role offers several advantages as it enables us to co-opt many useful methods from the field of software engineering. Due to the similarity of roles and class objects, one can adopt object oriented approach just as for class objects. For example, if a role is once codified, then reusability amounts to reassigning subjects to the role of the same responsibility. Similar duties can be easily constructed by modifying only a few attributes of an existing role.

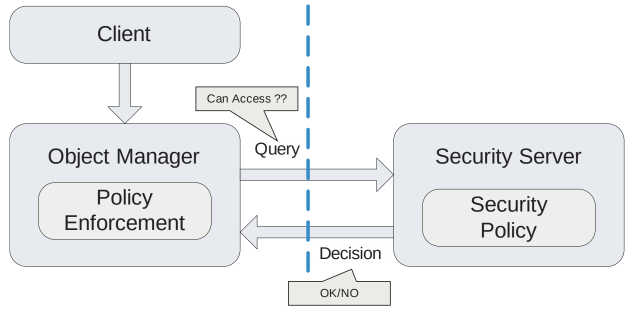

Figure 2: Security Server and Object Manager

2.2 A Secure OS: SELiunx

The Secure OS (Trusted Operating System) is an operating system which includes a security kernel providing protection from diverse threats. The security kernel approach assists in realizing the reference monitor with a trusted computing base (TCB) which enforces the security policy of a given system [7].

SELinux (Security-Enhanced Linux) is an instance of Secure OS. It is developed by National Security Agency (NSA), USA. SELinux implements the Flask architecture [9] to support several access control polices with high degree of flexibility in Linux OS. As for access control models, it provides RBAC and TE (Type Enforcement) [10]. The Flask architecture clearly divides logical access control policy and enforcement facility so as to enforce several types of access control. They implement Object Manager in the kernel subsystem to catch system actions ignited in the file system or network subsystem, Object Manager thus acts as an AEF (Access Enforcement Facility). Also, Security Server decides whether a given access request presented by Object Manager has to be granted or denied as a role of ADF (Access Decision Facility). Therefore, Object Manager can grant or deny the given access by the decision of Security Server [Fig. 2]. In Security Server, Access Vector Cache is included to reduce performance penalty by the enhanced access control. This kind of clear discrimination between access decision and enforcement gives a ability to change a currently enforcing security policy to the others in system runtime.

3 A Setuid Assigned Program Attack in Conventional Linux

3.1 Setuid (Set User ID)

In Unix or Linux, the setuid is used for a program which requires specific privileges to be executed properly. Though the executer is not the owner of the executing file, OS can give some privileges associated with the file owner's user id to the executer. For example, /bin/passwd program needs the privilege of root (uid=0) during changing the password (/etc/passwd). Thus while executing /bin/passwd program, the executing process has the right of root to change password file which is only accessible by the root.

The real credential of Linux kernel is associated with *euid* and *fsuid* field in kernel process structure. The fsuid is referenced whenever the file permission check is needed, and euid concerns the other permission check.

The setuid scheme offers privilege transition so that it can be abused if the executing program contains a runtime bug. In more detail, if a user runs setuid assigned program, the user process has privileges of the file owner during the execution. If the executing program has some bugs such as stack/heap overflow, format string, and the like, the user (attacker) may exploit the bug and try to run a shell program. If the exploitation is succeeded, the attacker enventually has a shell with file owner's privileges because the setuided level stays. Therefore, the attacker may illegally benefit from the privilege.

```
[kimhc@[redacted] kimhc]$ whoami
kimhc
[kimhc@[redacted] kimhc]$ ls -al ./apacheowned_exec
-rwsr-xr-x    1 apache   apache     13652 Aug 28 14:54 ./apacheowned_exec
[kimhc@[redacted] kimhc]$ ./b0f3xp
Using Address : bffffa88

[kimhc@[redacted] kimhc]$ ./apacheowned_exec $ADDR
sh-2.04$ whoami
apache
sh-2.04$
```

Figure 3: setuid program attack in generic Linux OS

3.2 Experiment I

Figure 3 shows our result of exploiting anexecutable file which has a buffer overflow bug. The owner of the file is a web service administrator (apache). In the result, an user (kimhc) attacks the executable file (apacheowned_exec) and spawns a shell with the privilege associated with that file (apache).

4 Privilege Transitional Attack in SELinux

4.1 Privilege Transition in SELinux

It is possible to exploit a Secure OS based on logical access control models if its configuration is inappropriate from the confusion of security administrator. SELinux enhance its security not from the architectural protection mechanism – we mean by hardware-aware based protection such as non-executable heaps – but from the logical access control models such as RBAC and TE. If a Secure OS offers any type of privilege transition in runtime, it is practicable to attack the system in a similar way in exploiting a setuid program in conventional Linux OS.

There are two way to trigger the transition of privileges in SELinux. One is Role Transition via *newrole* command – the role transition in policy configuration is deprecated in current version of SELinux – and the other is TE (Type) Transition. Through the TE transition, an executing program (process) can transit its privilege to the others.

TE transition makes new labels for the newly created subjects and objects based on TE configuration in security policy. Thus we can think that TE transition is a labeling decision mechanism based on policy configuration. When a subject or an object encounters the access control enforcement, the label associated with the corresponding access entities are referenced by the enforcement facility (Security Server). In a real enforcement, the label is identified as *sid* (Security Identifier). If a subject's sid is changed by the TE transition, then the subject can perform a new privilege associated with the transited sid. For example, if there

Table 1: A part of TE policy configuration in SELinux

type_transition sshd_t shell_exec_t:process user_t;
allow user_t httpd_admin_t:process transition; type_transition user_t www_exec_t:process httpd_admin_t;

exist a configuration as the first line of Table. 1, it means that a user shell process, executed through SSH, is firstly labeled as user_t type. Therefore, as the security context is changed from that of SSH program to a normal user shell program, the sid is also had to be changed.

The change of sid is dependent on the decision on the logical constitution of access control policy composed by security administrator. If a security administrator mis-configures the security policy due to the misunderstanding of system status or the high degree of complexity in access entity organization in the given system, there might be possible unwanted TE transitions.

4.2 Experiment II

As a setuid program in conventional Linux system, we configure a certain program to transit its type while executing [Tab. 1]. With this configuration, a user domain (user_t) can change its domain type to httpd_admin_t by the execution of a program of which domain is www_exec_t. Thus the process has rights of the web service administrator during the execution.

Figure 4: Privilege Transitional Attack in SELinux

In Figure 4, the result of *avc_enforcing* command, *enforcing*, means that a mandatory access control constituted by RBAC/TE configuration is currently activated beyond the DAC enforcement. A program, which contains a runtime bug and has the type www_exec_t, encounters an attack and the context is changed to root:user_r:httpd_admin_t, a privilege of web service administrator. Note that the root in this context is just a normal user in sense of RBAC, as it is assigned to the normal user role (user_r).

In SELinux, there is a command *runas* which supports Type transition according to security policy and it works via *execve_secure* security API function [Fig. 5]. If it is configured to transit from one context to the other context, the direct type transtion is possible with this command. Thus if an attacker analyze the mis-configured part of security policy, it is possible to transit using the command. In an experimental case [14], attackers tried to analyze and depicted the logical flow of configuration via *list_sids* command or direct trial of privilege test.

5 Discussion on Prevention

In last two sections, we examined that it is possible that a Secure OS, enforced with logical

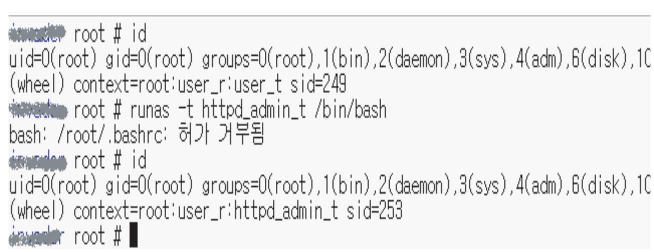

Figure 5: Privilege Transition via runas command

access control, can be harmed from intuitive and careless security configuration. Here we discuss the prevention issues on Secure OS.

There are two main perspectives to protect operating systems. One is the protection with the knowledge of system architecture. Making the position of stack in program runtime environment to be random or applying the non-executable stack are examples of such approach. On this approach, one has a thorough grasp of attack methods case by case and develops protection mechanisms in a given system.

However, there exist several methods to bypass these approaches. Already it is announced intruding techniques detouring Stack Guard, Stack Shield, and some of protection mechanisms of PaX. To the architecture-dependent approach, there may be continuous cycling process of developing patches for protection and trying to discover bypassing techniques.

Compared with the above approach, protection strategies by adopting logical access control, such as lattice-based or role-based access control, have advantages over the specific architectural concentration. For instance, the harm is limited within the role of exploited process in RBAC-based Secure OS and the attacker can not overturn the whole system. However, if access entities are too many so that security administrator can not grasp the whole understanding of the system, there might be possible mis-configuration thereby resulting in logical error in access control enforcement.

The architecture-dependent approach is still valid, but it is surely needed to inspect the logical flaw of the currently enforced policy for Secure OS which adopts the enhanced models of security. It is difficult to verify all the details of logical constitution of access control for security administrators. Therefore, the establishment of formal verification method has to be established with automatic and visual manner.

There are some instances of verification efforts for the case of RBAC such as Alloy [11], Petri Net [15], and Z [12]. DTOS [13] is the representative example of specification of Secure OS using Z. However, there are no tools connected with practical policy composition. Whenever a policy is changed, the policy enforcement tool has to verify the integrity of policy and shows the information flow to the security administrator analytically. In case of SELinux, a GUI-based configuration tool exists, but it is not yet adopt the formal verification techniques, thus security administrators still have burden for the confidence of security. Therefore, automatic verification tool, which can be used whenever the current policy has to be changed, has to be developed.

6 Concluding Remarks

This paper examined the privilege transitional attacks on generic Linux and SELinux, an instance of Secure OS, respectively. If the given Secure OS offers privilege transitional scheme in its enforcement rules, it is possible to exploit with the same degree of danger in conventional Linux OS. Therefore, it is strongly needed to connect the automatic and depictive formal verification tool with Secure OS by the time of policy alternation.

References

[1] Alphe One, "Smashing The Stack For Fun And Profit," Phrack Vol.7 Issue. 49, File 14 of 16, 1996.

[2] http://www.research.avayalabs.com/project/libsafe/

[3] http://www.immunix.org/stackguard.html

[4] http://www.angelfire.com/sk/stackshield/

[5] http://www.grsecurity.net/

[6] http://pageexec.virtualave.net/

[7] Edward G. Amoroso, "Fundamentals of Computer Security Technology" AT & T Bell Laboratories, Prentice-Hall PTR, 1994.

[8] D.F. Ferraiolo, J. Cugini, and D.R. Kuhn, "Role Based Access Control: Features and Motivations," In proc. of Annual Computer Security Applications Conference, IEEE Computer Sociery Press, 1995.

[9] Peter Loscocco, and Stephen Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," In Proc. of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX'01),Jun.2001. (http://www.nsa.gov/selinux/index.html)

[10] L. Badger, D.F Sterne, D.L Sherman, K.M Walker, and S.A. Haghighat, "Practical Domain and Type Enforcement for UNIX," In Proc. IEEE Symposium on Security and Privacy, pp. 66-77, May 1995.

[11] John Zao, Hoetech Wee, Jonathan Chu, and Daniel Jackson, "RBAC Schema Verification Using Lightweight Formal Model and Constraint Analysis," MIT Software Design Group Case Study, Dec 2002.

[12] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli, "Proposed NIST Standard for Role-Based Access Control Model," ACM Trans. on Information and Systems Security, Vol. 4, No. 3, pp. 224-274, Aug 2001.

[13] Secure Computing Corp., "DTOS Formal Security Policy Model," Technical Note of Secure Computing Corporation, 1996.

[14] Security Rsearch Group, In proc. of GIST(old K-JIST) Hacking Festival Workshop, Aug 2003.

[15] W. Shin, J.-G. Lee, H. K. Kim, K. Sakurai, "Procedural constraints in the extended RBAC and the coloured petri net modeling ," To be appered in IEICE Trans. on Fundamentals, Vol.E88-A, No.1, Jan. 2005