Article ID :1002— 1175(2004)01-0095-06

Administration of User Account in Secure OS *

ZHANG Xiang-Feng SUN Yu-Fang

(*Institute of Software, Chinese Academy of Sciences, Beijing* 100080, *China*)

(Received 23 December 2002 ; Revised 28 March 2003)

Abstract Many secure operating systems are developed based upon UNIX-like systems and many access control mechanisms and audit mechanism are introduced, but the system account file does not assure unique UID and might lead to confusion in audit trails. Users access rights in some security mechanisms are generally managed quite independently of account management and should also be deleted when one user is removed from the account file to avoid unintended reuse by another user. All those things require that the account file should be administrated in a way different from the traditional one in UNIX. Puts forward a mechanism to keep unique UID and to capture user account alteration in system call level. Puts the mechanism into practice in SLINUX, a variant of LINUX, and provide the performance analysis.

Key words secure OS, security mechanism, audit

CLC TP309

1 Introduction

With the fast development of the Internet, information system security is highly necessary. In an information system, the operating system is of great significance and acts as the basis of the whole information system, so the security of the operating system is the foundation of the whole information system security[1]. Many security mechanisms suchas mandatory access control, access control list and auditing are designed and deployed to secure the operating system. Audit mechanism is a necessary and supplementary measure of every secure information system and is regarded as the last defense line of the system.

Almost all security mechanisms, including mandatory access control (MAC) and access control list (ACL), are based on the subject ID because nearly all kinds of security attributes are bound to the subject IDs. Role-based access control (RBAC) seems a bit complex, but we could consider only the roles assigned to a user. As TCSEC[2] and CC[3] require, when a subject is removed from the system, all security attributes related to it should also be erased at the same time to avoid being re-used by another subject with the same ID (possibly introduced later). Most security mechanisms maintain security attributes in the kernel level to protect the security configurations and do not care about user account management. While most of the multi-user secure operating systems maintain user accounts separately from security attributes, it seems that user management has to be related to security attribute management in secure OS. But things might be complex in most operating systems.

*supported by the National 863 High-tech Program of China (863-306-ZD 12-14-2), the National Natural Science Foundation of China (60073022) and the Knowledge Innovation Engineering Program of the Chinese Academy of Sciences (KGCX 1-09)

The audit trail is probably the first thing a trained administrator will turn to when something goes wrong or something seems amiss[4]. An important thing is that the audit trail has to record all the security-related parameters of an event at the time the event occurs. Although the modifier "security-related" is a bit obscure, the most important one of those parameters is the subject's identity, that is, the user who is responsible for the event. In LINUX, a subject is generally a user (in reality a process is some kind of "direct" sponsor of events, but there must be a user who starts the process directly or indirectly and so the user becomes ultimately the real sponsor of all the events occurred in the process).

Unfortunately the user identification in many multi-user operating systems can be re-used and so un-unique. In UNIX-like systems, the shell command "useradd" can specify a new user's UID through an option "-u" no matter whether or not the UID has already been used by another user. Even we could be sure to assign a UID to only one user, a UID could still be shared by two or more users at different time (for example a deleted user's UID could be reused by another user). This might introduce confusion into the audit trail. So there should be some mechanism to avoid re-use of UIDs or to help tell apart users with the same UIDs.

We designed a secure operating system, SLINUX, a variant of LINUX, which incorporates several security mechanisms and hence incurs the two problems presented above. We introduced a mechanism to solve the problems. The performance test results given at the end of the paper show that the mechanism is workable.

2 Security mechanisms in SLINUX

There are various mechanisms that allow information systems protect the system resources and user resources, some of them are based on access control and some are based on information-flow control. Audit is different from them in that it is an after-the-fact mechanism. All these mechanisms can be separately put into practice and also be combined to further secure an information system.

Access control mechanisms are usually viewed in terms of an access control matrix (ACM), with rows representing active subjects (typically a user), columns representing passive objects (typically a file or a device or some kind of other resource) and cells showing the rights. Because storing the whole ACM would generally consume far too much space and there are always too many empty cells in ACM, real systems use either the rows or the columns of the matrix for access decisions. Implementations based on the rows attach a list of accessible objects to the subject and are named as "capability", while implementations based on the columns attach a list of subjects to the objects and are named as "access control list (ACL)"[5]. In SLINUX, ACL is implemented and the subject is identified with the user ID (The user name is rarely used except when a user is trying to login. The system kernel cares only about the user's UID and neglects the user's name thoroughly.). When an object (a file or a device) is deleted, its ACL is also erased from system kernel space at the same time.

Mandatory access control (MAC) is information-flow based. In MAC, each subject and/or object is assigned a security level, and the information flow is allowed if and only if the destination object has an equal or higher security level than that of the source subject or object. SLINUX implements two kinds of MAC, i.e. C-MAC and I-MAC. In C-MAC, the security level represents the confidentiality of an object or the highest security level of all objects that a subject can access, while in I-Mac, the security level stands for the integrity level of the object. In both C-MAC and I-MAC, a bit-set called "category" is attached to each security level and limits the access in a similar way as the security level does.

No matter what mechanisms are applied in an information system, there is always the possibility that someone could penetrate the system, especially from inside the organization owning the system. So it is undoubtedly necessary to audit the system. The audit mechanism can help review the access patterns and help discover malicious access. It can also act as a deterrent against perpetrator's habitual attempts to bypass the system security mechanisms. It supplies an additional form of assurance that all attempts to bypass the security mechanisms are recorded and can be traced. Even if the penetration succeeds, the audit trail will still provide assurance in accessing the damage done by the violation[6]. SLINUX introduces the audit mechanism as a security supplement to C-MAC, I-Mac and ACL. Audit rules may be based on IDs and security attributes of subjects and objects. To trace user actions and ascertain the responsibility, unique UIDs are required; otherwise accountability for audited actions might be more difficulty, though not impossible[7].

There are two common things for ACL, MAC and audit. First, users are identified by their UIDs and different users might have different ACL, security attributes and/or audit rules. To make administration easy and finally achieve a secure system, SLINUX requires that UIDs should be unique, just as audit mechanism does. In fact, several users sharing one UID is of little practical significance. Second, all the configuration for these mechanisms are kept and maintained in kernel memory and when a user is removed, both TCSEC and CC require to erase accordingly all its related access rights, security attributes and audit rules to avoid that they might be re-used later. So detection of user deletion is needed.

SLINUX introduces three special users, which can substitute for neither of the other two. They are system administrator (root), security officer (secoff) and audit administrator (auditor). All these users are added when installing the system and their names and UIDs should be kept unique and should never be deleted. Intuitively this also requires inspection on change of the account file's content.

3 Inspection of operations on account file

Seen from the above, there are three things we should do in SLINUX. First, we should keep unique UID in the account file; second, we should detection deletion of user in the account file, and lastly, we should ensure that special users are always in the account file. It seems that we could meet these requirements by modifying some tools already available. But things may not be like what they seem to be.

In UNIX-like systems, a user is identified with his name and identification number (UID), with only the UID is used in system kernel. The command "useradd" is used to add a user to the system. If the option "-u" is used with "useradd", the new user's UID can be specified at the command line; otherwise the command would automatically select a new UID for the new user. The command "userdel" can remove a user from the system. So it seems that we could modify the command "useradd" by disabling the "-u" option to avoid duplicated UIDs and modify "userdel" to catch the event of user deletion. Ensuring of the three special users could also be done in this way. SLINUX does the modification indeed, but this is not enough.

In fact, the system account file is a text file and can by default be read by anyone and written by only the system administrator (root). In SLINUX, the user "root" is no longer trusted because too many intrusions succeeded in other UNIX-like systems with the result that root privilege was stolen (The other too special users in SLINUX are introduced to partake root's privileges to secure the system). When the root privilege was stolen, the intruder can modify the account file at will. Generally speaking, an intruder would not use the available "useradd" or other similar tools to modify the account file. To do this, he might (1) edit

the file directly through any editor; or (2) replace the file with another one; or (3) find the location of the file on disk and modify disk blocks directly or (4) build his own tools. Direct access to physical devices is strictly limited by SLINUX in kernel but there are still too many ways to modify the account files, which means inspection on user actions to meet SLINUX s requirements seems a bit impractical.

4 Principle to acquire account alteration in SLINUX

All requirements of SLINUX on the account file are related to the content of the account file. Since inspection of user actions is impractical, we could turn to inspection the changes to content of the file.

For SLINUX limits access to physical devices in kernel (even root might be refused to do this), we could only care about system calls because system calls are the only way that all kinds of user applications get services from the operating system. Before an action that might alter the account file begins, the content of the file is saved; when the action is done, the content of the file is re-read (possibly not from disk due to cache buffer) and we could compare them to see if the action has done some alteration and what alteration to the file.

5 Implementation of account file inspection

In UNIX-like systems (including SLINUX), there are several system calls that can be used to modify a file, they are "open", "write", "truncate", "ftruncate", "rename", "unlink". Another system call, "mmap", might also be used to modify a file indirectly, but it requires that the file must be opened first, so we can just ignore it here.

Here "open" might cut a file to zero length with "O-TRUNC" flag; "write" can only follows opening of a file if the parameter "flags" for "open" does not include "O-RDONLY" flag; "ftruncate" acts just like "open" with "O-TRUNC" flag (but it can specify a new file length). So we could only check system call "open" without "O-RDONLY" flag and ignore "write" and "ftruncate".

Another thing to be considered is that a file could be opened by more than one process simultaneously and a process might open a file more than once at the same time (with different file descriptors), all of these "open"s are irrelevant to others. We could just treat them as separate "open"s of different files. To distinguish between multiple "open"s, we use <process-id, file-descriptor> pair to specify each "open".

So here is the way to acquire account alteration events. Before a file is to be opened, if the open flag includes "WRITE", test if the file s full name is that of the system account file or both files inode numbers are equal (in the case of hard link). If either test passes, save the content of the file to a buffer that is identified by <process-id, available-file-descriptor>. All the buffers of this kind are linked to system list. When a file is closed, we can check if the corresponding open flag contains "WRITE" and the file is the system account file. If both are yes, re-read the file and compare its content with the old copy saved in the buffer. Thus we can know if any user is added or deleted and if there are duplicated UIDs in the file.

The system call "truncate" seldom happens to cut a whole text line and we might regard a broken line (according to the format of the file) as a blank line. We can save the system account file into a buffer before truncating it. If the system call is done successfully, re-read the file s content and compare it with the saved

one.

There are two cases about the system call "rename". One is just simply to change a file's name to a new one (non-existent), and the other is similar to overwriting an existing file with another one, e.g. "mv -f /tmp/myfile /etc/passwd". As to the system call "unlink", we might think it as renaming a file to a non-existing file whose length is zero. When the system account file is overwritten by another file (maybe an old copy or even a binary file), we can acquire user alteration in the same way as "open"/"close" does. When the system account file is unlinked, all the users are deleted from the system.

6 Performance analysis

SLINUX implements several security mechanisms and we do account alteration detection based on SLINUX. We just give the performance difference before and after account alteration detection is introduced.

Four tasks are used to test performance difference. The first task repeats opening/closing the account file in read-only mode for 10000 times, but reads nothing from it. The second one repeats opening/closing the account file in write-only mode for 10000 times, but writes nothing to the file. The third one repeats opening the account file in append mode for 10000 times and appends a short string to the file each time. The last one is to build a new kernel from the source code. Tab. 1 shows the time each task costs.

Tab. 1　Results of performance test

	Seconds used with account detection disabled/T_0	Seconds used with account detection enabled/T_1	$[(T_1-T_0)/T_0]\times 100\%$
Task 1(read-open/close)	0.297025	0.308771	3.95%
Task 2(write-open/close)	0.488683	26.705513	5364.79%
Task 3(append/close)	0.625211	26.979326	4215.24%
Task 4(make kernel)	368.356030	371.142246	0.76%

From the table above, we can see that account alteration detection affects single operation on account file greatly (maybe the performance cost is unbearable) except the read-only mode, but as to the whole system, the performance cost is so little that it could be ignored thoroughly.

In fact, only very few system actions are related to the system account file, so the total cost to detect account alteration might be negligible. Hence, we can say that the mechanism we put forward in this paper to keep unique UID and to detect account alteration is practical.

7 Conclusion and future work

This paper states that the audit subsystem in a security operating system requires to keep account UID unique and to capture user account alteration. It also gives a solution. We introduce the solution to SLINUX, a variant of LINUX, and provide the performance analysis.

We do our work in system call layer and this might burden the system performance still a bit heavily. Future work includes bettering the code and improving the system performance further.

Maybe we should try other ways to acquire account alteration, e.g. to limit the access to the account file to some specific role (user root might possibly play the role), or to re-write related library functions completely. In fact, it seems a bit odd that Linux manages all kinds of objects (files, directories, devices, pipes,

IPCs, system clock, host name and so on) in kernel while leaving only user management to application layer thoroughly. Maybe it seems accordant to also put user management into kernel to achieve a uniform style.

References

[1] P Loscocco, S Smalley, P Muckelbauer, R Taylor, J Turner, J Farrell. The inevitability of failure: The flawed assumption of security in modern computing environments. In: Proceedings of the 21st National Information Systems Security Conference. 1998. 303—314

[2] National Computer Security Center. Department of defense trusted computer system evaluation criteria. DoD 5200.28-STD. 1985

[3] The International Organization for Standardization. Common criteria for information technology security evaluation——Part 1, 2, 3。1999

[4] Paul Whelan. Linux security auditing. Available at http://www.sans.org. 2001

[5] Deborah Downs, Jerzy Rub, Kenneth Kung, Carole Joran。Issues in discretionary access control。In:Proceedings of the 1985 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, 1985. 208

[6] Fort George G Meade. A guide to understanding audit in trusted systems. NCSC-TG-001. Version-2, Library No. S-228, 470. 1987

[7] Terry Escamilla. Intrusion detection: Network security beyond the firewall. Wiley Computer Publishing, 1998. ISBN 0-471-29000-9. 30

安全操作系统中用户账号的管理

张相锋 孙玉芳

(中国科学院软件研究所, 北京 100080)

摘 要 很多安全操作系统都是基于类 UNIX 系统开发的, 并按照 TCSEC 或 CC 的要求引入了强制访问控制和审计等安全机制, 但是并未保证用户账号的唯一性, 从而可能造成审计记录的混乱和用户权限的不正确重用, 这就要求改变原来的类 UNIX 系统的账号管理方式。提出了在系统调用层截取修改系统账号文件这类事件以保证用户 UID 唯一性的方案, 使得即使超级用户(包括通过成功的攻击而获取的超级用户权限)也无法任意修改用户账号数据库。这种机制已经在 SLINUX 系统中得到了实现。最后给出了该机制在SLINUX 系统上的性能测试结果。

关键词 安全操作系统, 安全机制, 审计