LSMPMON: Performance Evaluation Mechanism of LSM-based Secure OS

Toshihiro Yamauchi and Kenji Yamamoto

Graduate School of Natural Science and Technology, Okayama University,
3-1-1 Tsushima-naka, Kita-ku, Okayama, 700-8530 Japan
yamauchi@cs.okayama-u.ac.jp

Abstract

Security focused OS (Secure OS) is attracting attention as a method for minimizing damage caused by various intrusions. Secure OSs can restrict the damage due to an attack by using Mandatory Access Control (MAC). In some projects, secure OSs for Linux have been developed. In these OSs, different implementation methods have been adopted. However, there is no method for easily evaluating the performance of the secure OS in detail, and the relationship between the implementation method and the performance is not clear. The secure OS in Linux after version 2.6 has often been implemented by Linux Security Modules (LSM). Therefore, we determine the effect of introducing the secure OS on the performance of the OS by using the overhead measurement tool, the LSM Performance Monitor (LSMPMON). This paper reports the evaluation results of three secure OSs on Linux 2.6.36 by LSMPMON. The results show the effect of introducing the secure OS.

Keywords: *Performance Evaluation, Secure OS, Linux, LSM*

1. Introduction

It is very difficult to prevent all the attacks that occur when the weakness of a system is exploited. In addition, applying patches to compensate for vulnerabilities is not sufficient for preventing attackers from attacking computers. Therefore, secure OSs have attracted attention as a solution to these problems. A secure OS provides forced access control (Mandatory Access Control, MAC) and the minimum special privileges so that minimal damage occurs even if the root privilege is obtained by an attacker. The secure OS is based on the label system or path name system or others. There are several methods for implementing the secure OS, and the functions in these methods are different. Therefore, it is not easy to select a secure OS that is appropriate for use in the user's environment. In addition, the consequences of introducing a secure OS are not clear, and the change in the specifications and performance with the change of the version is difficult to determine.

After Linux 2.6, the function of the secure OS is implemented by a hooking system that calls a function group named Linux Security Modules (LSM) [1]. We paid attention to LSM, and we implement the performance evaluation mechanism of the LSM-based secure OS; this mechanism is named the LSM Performance Monitor (LSMPMON) [2],[3]. The LSMPMON records the processing time at the each hook point of LSM and the calling count. Therefore, we can evaluate the processing time for each hook and the point at which bottlenecks exist in the secure OS. The monitor enables us to easily compare the performances achieved by using secure OSs.

LSMPMON has been developed for evaluating LSM-based secure OSs. In this paper, a new version of LSMPMON developed for Linux 2.6.36 is described and results of the

evaluation of Security-Enhanced Linux (SELinux)[4][5], TOMOYO Linux [6][7], and LIDS [8], which are representative secure OSs in Linux, are presented. The unit of access control for resources in SELinux is label-based MAC, that in TOMOYO Linux is path-name-based MAC, and that in LIDS is i-node-based MAC. LSMPMON can be used to evaluate the influence of different methods of resource identification on performance. We evaluate the performance using benchmark software and report an analysis of the overhead incurred when a secure OS and each LSM hook are used on Linux 2.6.36. As a result, we clarify influence on performance by using the secure OS, and a characteristic by the difference of the identification method of resources and a change of the performance.

2. Security Focused OS

2.1. Evaluation of secure OS

Secure OS indicates OS that has the function to achieve MAC and the minimum privilege. In the secure OS, a security policy is enforced, according to which operations are limited to permitted operations that consume permitted resources; further, access control is implemented in the root privilege. Therefore, a secure OS can limit its operation, even if an attacker obtains root privileges via an unauthorized access.

In addition, by the secure OS, it can authorize every user and process to the minimum privilege. We evaluated the secure OS developed by using LSM that is implemented on Linux. In particular, we used SELinux, TOMOYO Linux, and LIDS. These secure OSs (refer to Figure 1) are used to explain the difference between the features and resource identification schemes. The resource identification methods used in the secure OS involve the use of a label, a path name, and an i-node number for management. In this example, the Web server is the target to be accessed, the path name is /usr/sbin/httpd, and the i-node number is 123, and the Web server is attached to the label called httpd_t. The resource is accessed as an object, the path name is /var/www/index.html, the i-node number is 456, and it is attached to the label called web_contents_t.

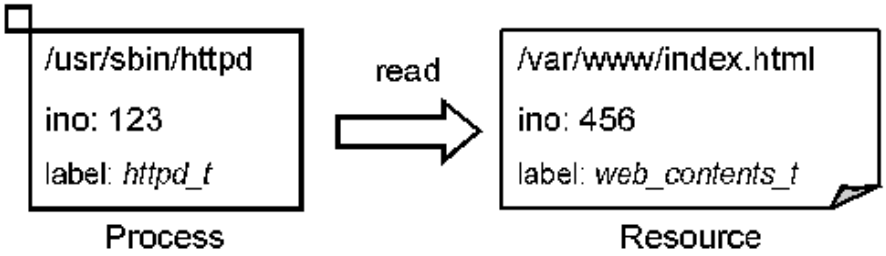

Figure 1. Process by which the Web Server Reads

TOMOYO Linux is included in a Linux kernel standard as a secure OS. In TOMOYO Linux, the path name base method is used for resource identification. In the example shown in Figure 1, it is understood that /usr/sbin/httpd reads /var/www/index.html. In addition, the identification process is based on knowledge of the path name and the execution history of the process.

LIDS uses the path name for setting the access control. On the other hand, it uses an i-node number internally in order to manage resources.

2.2. LSM

LSM is a function that defines the hook system calls for function group to the security check mechanism in the Linux kernel. A user can initially expand the security check function of the kernel by using this function. After Linux 2.6, the LSM is incorporated in a kernel, and the function of the secure OS is often implemented by the LSM. When LSM is valid, this is checking of the safety before accessing an object of the kernel inside by the callback function of LSM which is registered by user. The structure of the LSM is described below. When an AP invokes a system call, the DAC performs a security check. Next, a hook function of the registered LSM is called at a security checkpoint in the kernel, and a security check is performed by each secure OS. When the operation is approved by these checks, system call processing is performed, and access to resources is enabled. In addition, the security check is performed not only before system call processing but also during the system call processing.

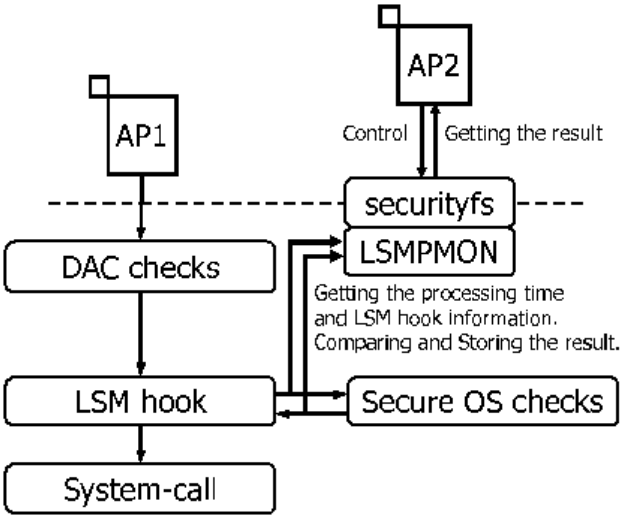

Figure 2. Structure of the LSMPMON

3. LSMPMON

3.1. Function

The main functions of LSMPMON are described below.

(1) Performance measurement of each LSM hook

Figure 2 shows the structure of the LSMPMON. The LSMPMON records the time before and after all the LSM hooks are called, in order to measure the processing time and calling count of the LSM hooks.

(2) Detection of the context switch

We prepare the flags that are used to detect context switches during the processing of the LSM hooks. The value of the flag can be checked in order to determine whether a context switch has occurred.

(3) Control using a simple interface

We use “securityfs” to simplify the user interface. Securityfs is a special virtual file system for security modules available in Linux 2.6.14. Securityfs is mounted on /sys/kernel/security, and it is used as an interface for controlling the secure OS and confirming the policy mainly. The module in which securityfs is used along with the LSM can easily perform data exchange in the user space and the kernel space without making original file system.

(4) Limited function of the measurement target

The manner in which a measurement target can be limited by using securityfs has been illustrated in (3). If the character string “s foo” is written for a specific file in securityfs, the LSMPMON stores a measurement result only when the subject name is “foo.” Similarly, If the character string “o bar” is written for a specific file in securityfs, the LSMPMON stores a measurement result only when the object name is “bar.”

(5) Function using the information of the system call

When an audit is effective, in the system call interface, the system call numbers are saved in the audit context structure, which is a member of the task structure of the process (AP1) that published the system call. When the LSMPMON monitor determines the processing time of the LSM hooks, it refers to audit context that is the member of the task structure of AP1 in order to acquire the system call number. For this reason, the system calls for each LSM will be able to measure the processing time, and thus, processing time can be saved.

3.2. Processing Flow of the LSMPMON

Processing performed by the LSMPMON is described below. First of all, AP invokes a system call. This time, if LSM hooks are called, rdtscll() function loads current Time Stamp Counter before LSM hooks is processed. Then actual processing of the LSM hook is done, Time Stamp Counter read again after the results came back. Then get the subject name and object name which performed access control and make a decision whether it is a measurement save object.

If it is a measurement object, confirmed the presence of context switches. If no context switch occurs, compare the processing time and the registered data of the processing time of same LSM hooks that same result in the access control. In that case, the shortest or longest processing time is saved the value. Furthermore, this processing time is added to the total processing time of the LSM hook that called in this time. Next, one adds the calling count that no context switch, and return to the original process. Finally, if a context switch occurs, one adds the calling count context switch. It will come back to the original process.

LSMPMON shows the results (min, max, ave, count, and cs_count of each hook); min is the shortest processing time, max is the longest processing time, ave is the average processing time for no context switch, count is the calling count without a context swich, cs_count is the calling count of the no context switch.

4. Evaluation of Secure OS using LSMPMON

4.1. Criteria in the Evaluation

We evaluated the secure OS on the basis of the following three criteria in order to determine the performance of the secure OS and effects of using different Linux kernels. We performed the evaluation on Linux 2.6.36 which is newer version than Linux used in [9].

(A) Effect of introducing the secure OS on performance

(B) A characteristic by the difference from the access control unit for resources in the secure OS

4.2. Evaluation Methods

We evaluated the file operation that the secure OS frequently performed access control processing. Contents of Evaluations are described below.

(A) Effect of performance by the secure OS

(Evaluation 1) By using the benchmark software LMbench[10], we measured the performance of each secure OS and evaluated the results in terms of the file operation. We show the effect of introducing the OS on performance.

(B) A characteristic by the difference from the access control unit for resources in the secure OS

Evaluation 1 does not indicate the time consumed by each LSM hook and the location of the bottleneck point. Therefore, we implemented the following steps:

(Evaluation 2) We measured the processing time corresponding to every LSM hook by using the LSMPMON in order to compare and evaluate these values. Thus, we clarify the effect of differences in the access control unit for resources on performance and show the characteristics of this effect.

(Evaluation 3) We compared every system call with the LSMPMON in detail. We measured the overhead of the LSM hooks in each system call by the LSMPMON, and we evaluated the results.

The evaluation environment was as follows: CPU, Pentium 4 (3.0 GHz); Memory, 1 GB; OS, Linux 2.6.36. In addition, all the measurements were obtained while running LMbench five times, and the results show the average processing time.

4.3. Evaluation of the Effect on Performance by the Introduction of the Secure OS

Table 1 shows the results of Evaluation 1. In this evaluation, LSMPMON was disabled. In SELinux, the rates of increase in the processing time of stat and open/close are high. In TOMOYO, the rate of increase in the processing time of open/close is high. In addition, in the other items of Table 1, the rate of increase in the processing time is comparatively high. Thus, it is thought that four processing involves a large overhead.

Table 1. Processing time of file operations and its increase rate in Linux kernel 2.6.36 measured by LMbench (unit: microsecond)

	Normal	SELinux		TOMOYO		LIDS	
stat	1.981	2.536	(28.0%)	1.556	(-21.5%)	2.119	(7.0%)
open/close	3.188	4.330	(35.8%)	4.051	(27.1%)	3.414	(7.1%)
Simple read	0.295	0.302	(2.3%)	0.289	(-2.0%)	0.307	(4.1%)
Simple write	0.250	0.257	(2.8%)	0.288	(15.2%)	0.263	(5.2%)

The processing time for the stat operation in TOMOYO Linux is the shortest among that in the three secure OSs. In addition, the rate of increase in processing time of the simple write is the highest among the rates for the three secure OSs.

The processing time of LIDS is comparatively shorter than the two other secure OSs. Therefore, it is thought that LIDS is the most suitable OS for file processing.

4.4. Features of Differences of Implementation

Table 2 shows the results of Evaluation 2. In this evaluation, LSMPMON was enabled in order to evaluate LSM hook functions. In addition, "N/A" in Table 2 indicates that the LSM hook functions are not implemented. LSM hooks based on i-node are used in SELinux and LIDS, while LSM hooks based on the path name are used in TOMOYO Linux.

Table 2. Processing time of LSM hooks called in each file operation (unit: microsecond)

hookname	SELinux	TOMOYO	LIDS
path_mknod	N/A	31.117	N/A
path_unlink	N/A	22.995	N/A
path_truncate	N/A	22.561	N/A
inode_init_security	19.878	0.067	0.067
inode_create	26.552	0.084	0.085
inode_unlink	0.368	0.081	0.241
inode_permission	0.191	0.036	0.089
dentry_open	0.392	N/A	N/A

In SELinux, the processing time of inode_init_security and inode_create is particularly long. This is because it is necessary to perform the recomputation of the label and to initialize each i-node that is newly created. In TOMOYO Linux, the processing time increases because of the function based on path name, such as path_mknod, path_unlink and path_truncate. It is thought that in order to check access permissions, it is necessary to compare of the character string and to obtain the path name.

In LIDS, there is no need to obtain the path name and to determine the label on LSM hooks. Thus, the total processing time consumed by the LSM hooks is short. It is thought that this is the factor because of which the overhead of the whole secure OS is small.

From the above evaluations, the following results were obtained:

(1) In SELinux, the overhead of open/close and create file is large. This is because labeling is necessary. Further, other items have a relatively large overhead.

(2) In the case of for TOMOYO Linux, where the path name is distinguished, the overhead is particularly large when the path name is obtained and deleted. In addition, the open/close operation is slow because it is necessary to reference the path name (compare the character string).

(3) LIDS has a small overhead in file processing because in LIDS, it is not necessary to perform labeling and to obtain the path name.

4.5. Detailed Evaluation of the Overhead in each System Call

Details of Evaluation 3 and evaluations are provided below.

(Evaluation 3-1) File open in SELinux (open system call)

(Evaluation 3-2) File open in TOMOYO Linux (open system call)

By using the system call information of LSMPMON, we evaluated the processing time required by LSM hooks (microsecond) for a system call. As in the past evaluation, we performed the evaluation by running LMbench five times.

Table 3 lists the results of Evaluation 3-1. Table 3 shows that inode_create and inode_init_security have a large overhead in open system call, the average of the processing time is large. In addition, inode_permission, inode_follow_link, file_alloc and file_receive have some overhead, because these hook functions are executed many times in open system call.

Table 3. Open System Call of SELinux (unit: microsecond)

hookname	sum	count	ave
capable	295432	494	0.199
cred_free	176165	519	0.113
d_instantiate	21610600	2300	3.132
file_alloc	323296698	458756	0.235
file_free	804890	1727	0.155
file_receive	327452825	457028	0.239
inode_alloc	2187320	1780	0.409
inode_create	55121407	692	26.552
inode_follow_link	269069736	445068	0.201
inode_free	63142	118	0.178
inode_init_security	41104746	692	19.800
inode_permission	1834565368	3174929	0.192
inode_setattr	253883	286	0.296
task_to_inode	45323	294	0.051
sum of average			46.990

Table 4 lists the results of the Evaluation 3-2. The main overheads in open system call of TOMOYO are path_mknod and path_truncate. Both path_mknod is LSM hooks for acquiring path name. In the kernel, the absolute pathname is obtained from the root directory by path_mknod. Therefore, the overhead of the LSM hook function increases. In addition, file_alloc, file_receive, inode_follow_link and inode_permission have some overhead, because these hook functions are executed many times in open system call.

The resource identification method of SELinux is different from that of TOMOYO. Therefore, SELinux does not use hook functions related to path name. On the other hand, TOMOYO use path_mknod and path_truncate, because TOMOYO use a path name for resource indetification.

Table 4. Open System Call of TOMOYO (unit: microsecond)

hookname	sum	count	ave
capable	94984	499	0.063
cred_free	109445	874	0.042
d_instantiate	628923	2455	0.085
file_alloc	43533453	406416	0.036
file_free	241676	1757	0.046
file_receive	2954862383	404643	2.434
inode_alloc	377088	1945	0.064
inode_create	144039	573	0.084
inode_follow_link	45345822	391823	0.038
inode_init_security	136912	678	0.067
inode_permission	306723466	2805134	0.036
inode_setattr	70220	258	0.091
path_mknod	63571961	681	31.117
path_truncate	18643343	231	26.902
sysctl	1125	4	0.094
task_to_inode	43407	260	0.055
sum of average			61.254

5. Conclusion

The secure OS in Linux is often implemented using an LSM. In this paper, a new version of LSMPMON developed for 2.6.36 is described, and the results of evaluation of three secure OSs is presented. We evaluated the LSM hooks function that is frequently used, especially during file operations. Below, we present the conclusion on the effect of introducing the secure OS introduction on performance, the features of the identification method of resources.

The following features are known as characteristics of the function of each secure OS: strict and strong security can be realized in SELinux, and a function for automatic learning of the policy is available in TOMOYO Linux.

On the other hand, the performance of these OSs has not been widely discusses. In embedded applications, the focus is on the secure OS, and it is important to evaluate the performance. Further, the effective environments change because of differences in the resource identification methods. In this article, we presented the above evaluation for an index in the introduction.

In addition, we showed that the LSMPMON could perform an accurate evaluation of the LSM hooks in every system call, analyze the performance of the secure OS, and determine the bottleneck points. Thus, we showed the utility of the LSMPMON. The source code of the LSMPMON has been published on the LSMPMON project page [2].

References

[1] C. Wright, C. Cowan, S. Smalley, J. Morris and G. Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel", Proceedings of 11th Annual USENIX Security Symposium, pp. 17-31, **(2002)**.

[2] LSM Performance Monitor, http://www.swlab.cs.okayama-u.ac.jp/lab/yamauchi/lsmpmon/.

[3] N. Matsuda, K. Satou, T. Tabata and S. Munetou, "Design and Implementation of Performance Evaluation Function of Secure OS Based on LSM", IEICE Transactions on Information and Systems, Vol. J92-D, No. 7, pp.963-974, **(2009)** (in Japanese).

[4] NSA, Security-Enhanced Linux, http://www.nsa.gov/selinux/.

[5] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System", Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, pp. 29-42 **(2001)**.

[6] TOMOYO Linux, http://tomoyo.sourceforge.jp/.

[7] T. Harada, T. Handa and Y. Itakura, "Design and Implementation of TOMOYO Linux", IPSJ Symposium Series Vol. 2009, No. 13, pp. 101-110, **(2009)** (in Japanese).

[8] LIDS, http://www.lids.org/.

[9] K. Yamamoto and T. Yamauchi, "Evaluation of Performance of Secure OS using Performance Evaluation Mechanism of LSM-based LSMPMON", 2010 International Conference on Security Technology (SecTech2010), Communications in Computer and Information Science (CCIS), vol. 122, pp. 57-67, **(2010)**.

[10] LMbench, http://www.bitmover.com/lmbench/.

Authors

Toshihiro Yamauchi

Received B.E., M.E. and Ph.D. degrees in computer science from Kyushu University, Japan in 1998, 2000 and 2002, respectively. In 2001 he was a Research Fellow of the Japan Society for the Promotion of Science. In 2002 he became a Research Associate in Faculty of Information Science and Electrical Engineering at Kyushu University. He has been serving as associate professor of Graduate School of Natural Science and Technology at Okayama University since 2005. His research interests include operating systems and computer security. He is a member of IPSJ, IEICE, ACM and USENIX.

Kenji Yamamoto

Received B.E. and M.E. degrees in computer science from Okayama University, Japan in 2009 and 2011, respectively. In 2011, he joined Hitachi Solutions, Ltd. His research interests include computer security.