Evaluation of Performance of Secure OS using Performance Evaluation Mechanism of LSM-based LSMPMON

Kenji Yamamoto and Toshihiro Yamauchi

Graduate School of Natural Science and Technology, Okayama University,
3-1-1 Tsushima-naka, Kita-ku, Okayama, 700-8530 Japan
yamamoto-k@swlab.cs.okayama-u.ac.jp, 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 OSes can restrict the damage due to an attack by using Mandatory Access Control (MAC). In some projects, secure OSes for Linux have been developed. In these OSes, 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, and a characteristic by the difference of the implementation method by using the overhead measurement tool, the LSM Performance Monitor (LSMPMON); the LSMPMON can be used to evaluate three different secure OSes.

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 OSes 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]. 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 OSes.

LSMPMON has been developed for evaluating LSM-based secure OSes. In this paper, a new version of LSMPMON developed for 2.6.30 is described and results of the evaluation of Security-Enhanced Linux (SELinux)[3][4], TOMOYO Linux [5][6], and LIDS [7], which are representative secure OSes 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 a benchmark software and report an analysis of the overhead incurred when a secure OS and each LSM hook are used. We perform the same evaluation for different versions of the kernel, namely, for the current kernel (2.6.30) and the old kernel (2.6.19), in order to verify the changes in performance. 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 at the version interval.

The contributions of this paper are as follows:
(1) In this paper, a new version of the LSMPMON is described. All evaluation methods for LSM-based secure OSes require the use of the LSMPMON. The LSMPMON can record the processing time and the calling count of each LSM hook. The results are useful for analyzing the performance of secure OSes.
(2) In this paper, first, the difference between different secure OSes is reported. The reports are based on the evaluation results obtained by using LSMPMON. The reports clarify the relation between the access control method and the performance of secure OSes.
(3) The difference between different kernel versions of Linux is described. These results reveal that the performance of the secure OS strongly depends on the kernel version.

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

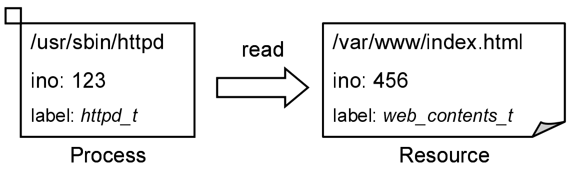

Fig. 1. Process by which the Web server reads a Web page

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 OSes (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.

SELinux is included in the Linux kernel standard as a secure OS. In SELinux, offers increased safety since it is based on TE (Type Enforcement), MAC, and RBAC (Role-Based Access Control). In SELinux, the label base method is used for resource identification and control by assigning a label, which is called a domain, and attaching a type to a process or a file. In the example shown in Figure 1, it can be seen that the process in which a domain called httpd_t was added reads the file that was assigned a type called web_contents_t.

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 Linux2.6, 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

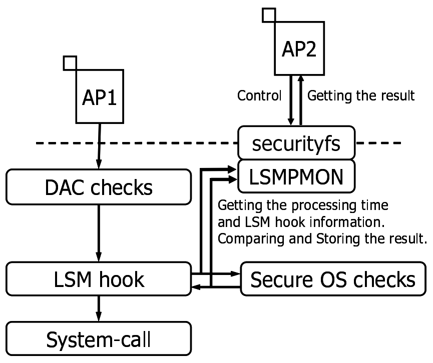

Fig. 2. Structure of the LSMPMON

function of the registered LSM is called at a security checkpoint (Linux 2.6.30, 186 points) 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.

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.

Figure 3 shows an example LSMPMON behavior. Transferring data between kernel space and user space such that data is transmitted to an existing AP and performed, by using securityfs.
(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

```
  Enable LSMPMON
% echo 1 > /sys/kernel/security/lsmpmon/control

  Disable LSMPMON
% echo 0 > /sys/kernel/security/lsmpmon/control

  Show the result
% cat /sys/kernel/security/lsmpmon/result
hook           min   max    ave  count    cs_count
----------------------------------------------------
         .
         .
         .
inode_create   97    67103  105  2725605   0
inode_link
inode_unlink   97    72728  114  2725600   0
         .
         .
         .
```

Fig. 3. A behavior example of the LSMPMON

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 caling 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.

In the behavior example in Figure 3, 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.
(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
(C) Changes in the performance and specifications across different versions

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[8], 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.

(C) Changes in performance and specifications across versions
(Evaluation 3) We perform similar evaluation in the Linux kernel of the version that is different from that in Evaluation 1; then, we compare the results and consider the change in the performance of each secure OS. However, by evaluation in LMbench alone, we cannot understand the factors that cause the change in the performance and determine the bottleneck point. Therefore, in order to determine the point at which the performance changed in this evaluation, we implemented the following steps:
(Evaluation 4) 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 compared the results for different versions and evaluated the results. Therefore, we compare the overhead in detail and show the change in the performance and specifications across versions. The evaluation environment

Table 1. Compare to each secure OS in the new kernel

	SELinux	TOMOYO Linux	LIDS
Use of LSM hooks	154	14	46
Identification method of the resource	label	path-name + execution history	inode
Version	3.6.12-39.fc11	2.2.0	2.2.3rc8

Table 2. Compare to each secure OS in the old kernel

	SELinux	TOMOYO Linux	LIDS
Use of LSM hooks	149	17	38
Identification method of the resource	label	path-name + execution history	inode
Version	2.4.6-80.fc6	2.0	2.2.3rc1

was as follows: CPU, Pentium 4 (3.0 GHz); Memory, 1 GB; OS, Linux 2.6.30.4 (new kernel) and Linux 2.6.19.7 (old kernel) is. In addition, all the measurements were obtained while running LMbench five times, and the results show the average processing time. The identification method for each secure OS and the object that performs access control, the calling count of LSM hooks, and the Secure OS version are shown in Table 1 and Table 2.

4.3 Evaluation of the effect on performance by the introduction of the secure OS

Table 3 shows the results of Evaluation 1. In SELinux, the rates of increase in the processing time of stat, read/write, and 0K file creation are the highest among the rates for the three secure OSes. In addition, in the other items of Table 3, the rate of increase in the processing time is comparatively high. Thus, it is thought that file processing involves a large overhead.

The processing time for the stat operation in TOMOYO Linux is the shortest among that in the three secure OSes. On the other hand, the file creation and deletion in this are much slower in this kernel than in the old kernel. Further, a large overhead is incurred in open/close. Thus, the overhead in particular may become large as the email server repeats the creation of a file.

The processing time of file generation and deletion in LIDS is shorter than in the two other secure OSes. Thus, the rate of increase in the processing time is small in the other items of Table 3. Therefore, it is thought that LIDS is the most suitable OS for file processing.

4.4 Features of differences of implementation

Table 4 shows the results of Evaluation 2. In addition, "N/A" in Table 4 indicates that the LSM hooks 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.

In SELinux, the processing time of inode_create and inode_init_security 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,

Table 3. Processing time of file operations and its increase rate in Linux kernel 2.6.30 measured by LMbench (unit:μs)

	Normal	SELinux	TOMOYO	LIDS
stat	1.79	2.67 (48%)	1.83 (2%)	2.20 (22%)
open/close	2.74	3.94 (44%)	4.78 (75%)	3.28 (20%)
read/write	0.37	0.47 (29%)	0.37 (0%)	0.37 (0%)
0Kfile Create	15.16	58.18 (283%)	53.58 (253%)	16.84 (11%)
0Kfile Delete	8.20	9.26 (12%)	33.10 (303%)	9.91 (21%)
10Kfile Create	47.84	89.38 (87%)	83.32 (74%)	48.42 (1%)
10Kfile Delete	20.06	20.18 (0.6%)	42.68 (112%)	21.16 (5%)

Table 4. Processing time of LSM hooks called in each file operation (unit:μs)

hookname	No_secure	SELinux	TOMOYO	LIDS
path_mknod	0.045	N/A	13.609	N/A
path_unlink	0.035	N/A	23.164	N/A
path_truncate	0.060	N/A	13.426	N/A
inode_init_security	0.036	20.658	N/A	N/A
inode_create	0.035	19.037	N/A	N/A
inode_unlink	0.036	0.216	N/A	0.127
inode_permission	0.035	0.159	N/A	0.192
dentry_open	0.041	0.222	17.381	N/A

such as path_mknod and path_unlink. 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 creating files 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 Differences between performance and specifications of different kernel versions

Comparison of performance of different kernel versions

Table 5 lists the results of the Evaluation 3. In SELinux, the performance of current kernel is considerably better than that of the old kernel in almost all aspects. In particular, the rate of increase in the processing time for read/write reduced from 157% to 29%. Similarly, the rate of increase in the processing time for 0K file delete reduced from 80% to 12%. However, the rate of increase in the processing time for stat, read/write, and 0K file create are still highest among the three secure OSes.

Table 5. Processing time of file operations and its increase rate in Linux kernel 2.6.19 measured by LMbench (unit:μs)

	Normal	SELinux	TOMOYO	LIDS
stat	2.64	4.34 (64%)	2.63 (0%)	2.79 (6%)
open/close	3.98	6.45 (62%)	9.33 (134%)	3.96 (0%)
read/write	0.58	1.49 (157%)	0.58 (-1%)	0.58 (-1%)
0Kfile Create	11.76	42.80 (264%)	16.36 (39%)	14.00 (19%)
0Kfile Delete	6.25	11.22 (80%)	9.61 (54%)	6.67 (7%)
10Kfile Create	34.34	69.26 (102%)	37.78 (10%)	35.48 (3%)
10Kfile Delete	16.14	19.28 (19%)	19.02 (18%)	16.90 (5%)

In TOMOYO Linux, the performance of file creation and deletion of current kernel has greatly deteriorated compared to that of the old kernel. On the other hand, the rate of increase in the processing time for open/close has reduced.

Compared to the previous LIDS kernel, the processing time has increased for many items. However, there is not the item where processing time extremely increases. In addition, the processing time for file creation and deletion is considerably shorter than the corresponding time required for the other two secure OS's.

Detailed evaluation of the overhead in each system call

On the basis of the results of Evaluation 3, we compared old and new kernel to measure its performance in detail, especially in the case of items with respect to which the performance of the old and new kernel differ greatly. Details of Evaluation 4 and evaluations are provided below.
(Evaluation 4-1) File deletion in SELinux (close system call)
(Evaluation 4-2) File creation in TOMOYO Linux (creat system call)

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

Table 6 lists the results of Evaluation 4-1 A comparison of the new kernel with the old kernel shows that the processing time of LSM hooks has increased in the latter for many items. However, the processing time for sock_rcv_skb hook has significantly reduced. In the old kernel, this LSM hook has an especially large overhead. In the new kernel, the processing time of LSM hooks has decreased, resulting in the reduction of the overall processing time.

Table 7 lists the results of the Evaluation 4-2. The main overheads in the old kernel are path_mknod and dentry_open, and the main overheads in the new kernel is inode_create. Both inode_create and path_mknod are LSM hooks for acquiring path name. In the case of the old kernel, inode_create can also acquire a relative path, which had used its own implementation of TOMOYO Linux, to acquire the absolute path. In the new kernel, the absolute pathname is obtained from the root directory by path_mknod. Therefore, the overhead of the LSM hook function increases. In addition, dentry_open is an LSM hook that is used to reduce the number of times policy search has to be carried out; however, this function in itself results in a large overhead. The change in the performance of the LSM hooks results in a large overhead in the case of creat system call. We arrive at the following conclusions on the basis of the above-mentioned evalua-

Table 6. Close system call in defferent version of SELinux kernel (unit:μs)

2.6.30				2.6.19			
hookname	sum	count	ave	hookname	sum	count	ave
file_free	183043.630	1370941	0.133	file_free	166246.612	1591046	0.104
inode_free	2105.578	10458	0.201	inode_free	2380.437	14669	0.162
inode_delete	97.809	973	0.100	inode_delete	62.058	1074	0.058
sock_rcv_skb	989.218	8253	0.120	sock_rcv_skb	11027.765	14583	0.756
sk_free	1617.659	5821	0.278	task_free	3.907	31	0.126
cred_free	10.074	92	0.109				
		average sum	0.942			average sum	1.206

tion results.
(1) In SELinux, there is a decrease in the overhead for file deletion because of a decrease in the rate of increase in the processing time of specific LSM hooks. In addition, it may be said that rate of increase in the processing time for all items except file creation decreased, and the performance improved.
(2) In TOMOYO Linux, the rate of increase in the processing time for the generation and deletion of a file is very large. This is because the function of obtaining the path name that is a unit of access control, from its own implementation for changes the implementation that using LSM hooks. In addition, the rate of increase in the processing time for open/close decreased.
(3) In LIDS, the rate of increase in processing time has increased in general; however, in the case of file generation and the deletion, rate of increase in processing time is the still smallest among the three OS's.

5 Conclusion

The secure OS in Linux is often implemented using an LSM. In this paper, a new version of LSMPMON developed for 2.6.30 is described, and the results of evaluation of three secure OSes 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, and the change in the performance across different versions of each secure OS.

In SELinux, the performance improves in many items. However, the overhead is large in many file operations that involve file creation, etc. The main factor affecting the overhead of file creation is the determination and initialization of the label because it using a label-based method.

The overhead of the file creation and file deletion, for which it is necessary to obtain and delete of the path name, is large because TOMOYO Linux controls it by a path-name-based method. In addition, the overhead of the open/close operation is comparatively large because it is necessary to reference the path name (compare the character string).

The overhead of file creation and file deletion in LIDS is small compared to that in the two other methods because LIDS is an i-node-based method, and it is not necessary to determine a label and obtain of the pass-name. In addition, the overhead in the other file operations is comparatively small.

Table 7. Creat system call in different version of TOMOYO Linux kernel (unit:μs)

2.6.30				2.6.19			
hookname	sum	count	ave	hookname	sum	count	ave
inode_permission	127585.395	3588000	0.035	inode_permission	366962.831	8124000	0.045
file_alloc	31367.404	897000	0.035	file_alloc	77799.981	2031000	0.038
inode_create	31240.724	897000	0.035	inode_create	5971299.965	2031000	2.940
inode_alloc	31425.973	897000	0.035	inode_alloc	72487.431	2031000	0.036
inode_init_security	33046.422	897000	0.035	inode_init_security	74019.938	2031000	0.036
d_instantiate	32385.181	898000	0.036	d_instantiate	72418.537	2031000	0.035
path_mknod	12207508.304	897000	13.609	task_free	2.593	8	0.324
dentry_open	22372903.197	897000	24.942	task_kill	0.402	2	0.201
cred_free	0.138	2	0.069				
		average sum	38.830			average sum	3.655

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 OSes 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 vary 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[9].

References

1. Wright, C., Cowan, C., Smalley, S., Morris, J., Kroah-Hartman, G.: Linux Security Modules: General Security Support for the Linux Kernel. Proceedings of 11th Annual USENIX Security Symposium, pp. 17-31 (2002).
2. Matsuda, N., Satou, K., Tabata, T., Munetou, S.: Design and Implementation of Performance Evaluation Function of Secure OS Based on LSM. The Institute of Electronics,Information and Communication Engineers Trans. Vol.J92-D, No.7, pp.963–974, 2009.
3. NSA, Security-Enhanced Linux, http://www.nsa.gov/selinux/
4. 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, pp. 29-42 (2001).
5. TOMOYO Linux, http://tomoyo.sourceforge.jp/
6. Harada, T., Handa, T., Itakura, Y.: Design and Implementation of TOMOYO Linux. IPSJ Symposium Series Vol. 2009, No. 13, pp. 101-110 (2009).
7. LIDS, http://www.lids.org/
8. LMbench, http://www.bitmover.com/lmbench/
9. LSM Performance Monitor, http://www.swlab.cs.okayama-u.ac.jp/lab/yamauchi/lsmpmon/