Electronic Preprint for Journal of Information Processing Vol.23 No.5

Regular Paper

Reducing Resource Consumption of SELinux for Embedded Systems with Contributions to Open-Source Ecosystems

Yuichi Nakamura[1,2,†1,a)] Yoshiki Sameshima[1,b)] Toshihiro Yamauchi[2,c)]

Received: November 27, 2014, Accepted: June 5, 2015

Abstract: Security-Enhanced Linux (SELinux) is a useful countermeasure for resisting security threats to embedded systems, because of its effectiveness against zero-day attacks. Furthermore, it can generally mitigate attacks without the application of security patches. However, the combined resource requirements of the SELinux kernel, userland, and the security policy reduce the performance of resource-constrained embedded systems. SELinux requires tuning, and modified code should be provided to the open-source software (OSS) community to receive value from its ecosystem. In this paper, we propose an embedded SELinux with reduced resource requirements, using code modifications that are acceptable to the OSS community. Resource usage is reduced by employing three techniques. First, the Linux kernel is tuned to reduce CPU overhead and memory usage. Second, unnecessary code is removed from userland libraries and commands. Third, security policy size is reduced with a policy-writing tool. To facilitate acceptance by the OSS community, build flags can be used to bypass modified code, such that it will not affect existing features; moreover, side effects of the modified code are carefully measured. Embedded SELinux is evaluated using an evaluation board targeted for M2M gateway, and benchmark results show that its read/write overhead is almost negligible. SELinux's file space requirements are approximately 200 Kbytes, and memory usage is approximately 500 Kbytes; these account for approximately 1% of the evaluation board's respective flash ROM and RAM capacity . Moreover, the modifications did not result in any adverse side effects. The modified code was submitted to the OSS community along with the evaluation results, and was successfully merged into the community code.

Keywords: SELinux, open-source software (OSS), embedded system

1. Introduction

M2M (Machine-to-Machine) technologies are widely used in various fields such as utilities, manufacturing , transportation, and health care [1]. As a result, embedded systems are increasingly connected to the Internet. One of the most typical embedded systems connected to the Internet is an M2M gateway, which bridges sensor nodes without IP addresses to the Internet [2]. Linux is a popular OS for those embedded systems, because development environments are available as freeware and actively maintained by the open-source software (OSS) ecosystem. Security for devices that run embedded Linux is an important issue, because those devices are typically connected to the Internet. Once vulnerabilities are exploited by attackers, they can destroy a system, steal information, and attack other systems. For PC servers, applying security patches and anti-virus software are effective countermeasures. However, these measures are impractical for embedded systems. Applying security patches to an embedded system is difficult for two reasons. First, developing patches is difficult because device vendors often customize embedded Linux systems, using them in place of standard Linux distributions. Device-specific security patches are not typically provided by Linux distributions. Second, installing modified programs can be difficult. Platform software is often stored in read-only flash ROM file systems. For firmware updates, it is necessary to rewrite the entire file system to install modified programs. However, firmware updates are risky because the device will not be usable if the update fails. It is also difficult to use anti-virus software. Preparing pattern files is diffcult because embedded Linux system configurations differ depending on the type of device. In addition, updating pattern files requires a risky firmware update.

Address space layout randomization (ASLR) and access control are effective security technologies not only for PC servers but also for embedded Linux systems. ASLR protects against attacks that exploit memory management vulnerabilities by randomizing stack and heap layouts, and an implementation for embedded systems is available [3]. However, ASLR only protects against attacks that exploit memory management vulnerabilities, and techniques to bypass it have been reported [4], [5]. Access control blocks attacks by employing a permission check. Linux provides access control based on file permissions. However, if an attacker can obtain root privileges, which provide access to all commands and files then no security technique will be effective.

Mandatory Access Control (MAC) is useful for limiting root

[1] Hitachi Solutions, Ltd., Shinagawa, Tokyo 140–0002, Japan
[2] Okayama University, Okayama, 700–0082, Japan
[†1] Presently with Hitachi, Ltd.
[a)] yuichi.nakamura.fe@hitachi.com
[b)] yoshiki.sameshima.vf@hitachi-solutions.com
[c)] yamauchi@cs.okayama-u.ac.jp

privileges [6]. With MAC, access to a resource is checked based on a set of rules called the security policy, and no user, including root, can avoid the check. If attackers obtain root privileges, their behavior can still be confined by MAC security policy, and attack attempts will fail owing to lack of access rights. The Linux kernel has a framework called Linux Security Modules (LSM) [7] to implement MAC. On top of LSM, various MAC systems such as SELinux [8], TOMOYO [9] and SubDomain [10] have been developed and are used in PC server systems. They are also useful for embedded systems, because security policy is built in at implementation time, therefore updates are not necessary.

In LSM-based MAC systems, SELinux is the most widely used and actively maintained implementation. SELinux provides the most fine-grained access control model based on permission and type enforcement (TE) [11], and has been researched since the 1980's.

However, to use SELinux on embedded systems, there are resource usage issues, and modified code must receive feedback from the SELinux community. SELinux developers have primarily focused on PC server implementations, and many features have been added. As a result, resource consumption has become unacceptable for embedded systems. To reduce resource consumption, Linux kernel and related OSS code must be modified. In addition, modified code must be merged into code maintained by OSS communities to ensure its long-term use. OSS code is made public by various open-source licenses; developers from around the world continuously submit patches, which are reviewed and merged into the OSS community code according to their development processes [12]. If modified code is not merged into the OSS community code, the developer must correct the modified code for every version up to the most current version of the target OSS. Conversely, if modifications are merged, they are maintained by OSS communities and the modified code will be used for an extended period.

We propose Embedded SELinux, in which SELinux resource usage is reduced with code modifications that are acceptable to the OSS community. Embedded SELinux is mainly targeted for embedded devices directly connected to the Internet and whose resource is constrained to save energy and cost. One of the most typical device is an M2M gateway, where CPU clock is often 200–600 mhz and RAM size is 32–128 MBytes. Consumer Electronics (CE) devices (Low-end TV, Set Top Boxes and In Home Displays) are also example of such devices. For these devices, resource usage is reduced using three techniques. First, the Linux kernel is tuned to reduce CPU overhead and memory usage. Second, unnecessary code is removed from userland libraries and commands. Third, security policy size is reduced by our policy-writing tool. In addition, to be acceptable to the OSS community, the side effects of the modified code are measured, and removed code can be selected by a flag at build time. We implemented and evaluated our system on an embedded system. The result shows that resource consumption is acceptable for an embedded system. Code modifications were proposed to the OSS community, and eventually merged into the community code.

In summary, we provide the following contributions.

- We determined that code modifications will be necessary to resolve the SELinux resource consumption problem on embedded devices. In particular, the overhead of the SELinux security check, memory consumption, and file size increases are not acceptable for embedded devices. In addition, we clarify the requirements for modifications to existing code that will be contributed to the OSS ecosystem.
- We built an embedded SELinux implementation that reduces resource consumption with modifications that are acceptable to the OSS community.
- We demonstrate the effectiveness of embedded SELinux without side effects on an embedded device evaluation board mainly targeted for M2M gateway and CE devices. Modifications were also contributed to the OSS community code and successfully merged. As a result, our work creates the basic structure of SELinux applications on embedded systems, which is used by other works such as SEAndroid [13] and Buildroot [14].

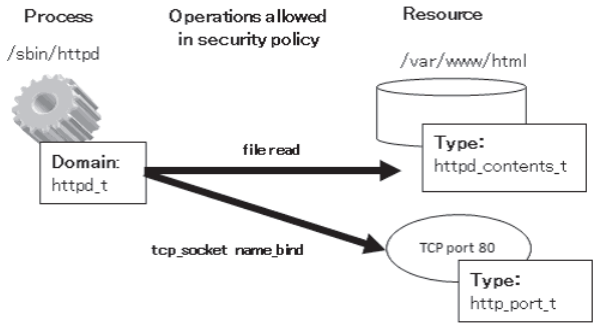

Fig. 1 Main feature of SELinux: TE (Type Enforcement).

2. Problems in Applying SELinux to Embedded Systems

After an overview of SELinux, this section describes the resource usage problems of SELinux for embedded systems, and the requirements for tuning code to facilitate its acceptance by the OSS community.

2.1 Overview of SELinux

The primary feature of SELinux is a MAC referred to as type enforcement (TE). An overview is shown in **Fig. 1**. Processes are identified by labels called *domain*, and resources are identified by labels called *type*. By default, a domain cannot access types. A domain is only able to access types that are explicitly specified in the security policy. In the example shown in Fig. 1, the label *httpd_t* t is assigned to the http daemon process, the label *http_content_t* is assigned to /var/www, and the label *http_port_t* is assigned to TCP port 80. In the security policy, http_t is granted permission related to file reading and port binding. As a result, the http daemon is allowed to read /var/www and bind TCP port 80; no other access is permitted, unless described in the policy.

These access control functions are implemented in the Linux kernel using LSM hook functions to perform security checks during system calls. In addition, a userland program is necessary to manage security policy and labels. For example, when the security policy is changed, the kernel must reload it.

Table 1 System call execution time (unit: micro second) on an embedded system measured by lmbench and calculated overhead of SELinux.Overhead is rate of increase in execution time from SELinux disabled kernel.

lmbench	SELinux disabled	SELinux	
Null read	2.39	5.49	(130.0%)
Null write	2.07	5.10	(146.6%)
Stat	21.48	42.29	(96.9%)
Create	108.18	284.98	(163.4%)
Unlink	67.74	126.3	(86.4%)
Open/close	32.55	62.82	(93.0%)
Pipe	33.56	55.96	(66.8%)
UNIX socket	76.12	99.80	(31.1%)
TCP	259.52	316.58	(22.0%)
UDP	162.76	207.85	(27.7%)

Table 2 read/write execution time on an embedded system (unit: micro second), measured by Unixbench and SELinux overhead.

Unixbench read/write	SELinux disabled	SELinux	
256 byte read	7.95	13.25	(66.6%)
256 byte write	12.84	21.42	(66.8%)
1,024 byte read	12.79	17.97	(40.5%)
1,024 byte write	19.25	27.69	(43.9%)
4,096 byte read	33.96	39.45	(16.2%)
4,096 byte write[*1]	806.45	781.25	(−3.1%)

2.2 Resource Usage

To use SELinux, extra features must be enabled in the Linux kernel and userland must be added to the standard Linux system. These increase system call overhead, file size, and memory consumption. To reduce power and costs, the hardware resources of embedded systems are significantly more constrained than PC server systems. For example, in the case of a Linux based M2M gateway, the CPU clock speed is approximately 200 MHz to 600 MHz, the architecture utilizes ARM and SH, RAM size is around 64 MBytes, and flash memory is used for storage. As a result, the resource usage of SELinux is not acceptable for such embedded systems.

2.2.1 Overhead in System Calls

When SELinux is enabled, there is overhead for system calls to check the security policy. SELinux is implemented based on Flux Advanced Security Kernel (FLASK) architecture [15], where such overhead is reduced by a cache mechanism called Access Vector Cache (AVC). In a PC environment, Loscocco, P. et al. [8] measured this overhead and concluded that it was insignificant. However, problems are encountered when using SELinux on an embedded system. An example of SELinux overhead measured on an embedded system is shown in **Tables 1** and **2**. These measurements were performed using lmbench [16] and Unix Bench [17]. The embedded system platform was composed of a SH7751R (SH4 architecture, 240 MHz) processor, Linux 2.6.22 and SELinux security policy whose size is 60 Kbytes and number of rule is just 2,000. In particular, read/write overhead is a problem, because they are executed frequently and the overhead is significant. Overhead of greater than 100% was observed during null reads/writes. Moreover, 16% of the overhead remained when reading a 4,096 buffer. A size of 4,096 bytes is often used for I/O buffers, because it represents the page size for many CPUs in embedded system architectures such as SH and ARM.

[*1] Performance is much worse than others because filesystem cache is fully occupied. Cache is working in others.

Table 3 Files related to SELinux.

Component	Additional features
Kernel	The SELinux access control feature, audit, and xattr support in filesystem.
Library	libselinux,libsepol, and libsemanage
Command	Commands to manage SELinux such as load_policy. Additional options for existing commands, such as -Z option for ls to view file label.
Policy file	The security policy

2.2.2 File Size Increase

The kernel and userland file sizes increase when SELinux is ported, because of the components listed in **Table 3**. The increase is approximately 2 MBytes if SELinux for PCs (the SELinux included in Fedora Core 6) is ported without tuning. However, this increase is not acceptable for embedded systems, because flash ROM with a capacity of less than 32 MBytes is often used to store the file system. If SELinux consumes 2 MBytes then this is considered excessive.

2.2.3 Memory Consumption

SELinux has data structures in the kernel to load the security policy. On a PC, the memory consumed by the security policy is approximately 5 MBytes. However, this is also unacceptable for embedded systems, because the RAM capacity is often less than 64 MBytes and swapping is not effective. If SELinux is used for embedded systems, the possibility that memory can not be allocated increases. If memory can not be allocated, applications will not work correctly.

2.3 Acceptability to the OSS Community

Code must be modified to reduce the resource usage described above. Modified code must be merged to the related OSS community source tree to obtain benefits from the OSS ecosystem. If not merged, modifications must be ported for every version up through the target OSS. To be merged, modifications must be accepted by the target OSS community. There are two primary requirements for acceptance. First, the modifications must not affect existing functions. Second, the modifications cannot cause side effects related to performance.

3. Overall Approach of Embedded SELinux

To apply SELinux to embedded systems, we propose embedded SELinux. The approach for using embedded SELinux to resolve the problems described in the previous section is described in the following section.

3.1 Reducing Resource Usage

The resource usage of SELinux shown in Table 3 is reduced as follows.

3.1.1 Kernel

The overhead related to system calls, RAM, and file size usage are all candidates for tuning. To mitigate system call overhead, we focus on reducing overhead in reads/writes, because overhead is significant as shown in Section 2.2.1. To reduce RAM and file size usage, the simplest method to reduce resource usage is to remove unused features. However, it is difficult to remove features from the kernel, because this will significantly affect existing fea-

Table 4 Features included in SELinux userland. Features 1,2, and 3 are used for embedded SELinux.

#	Feature	Related packages
1	Load policy Load security policy file to the kernel	libselinux
2	Change labels View and change domains and types	libselinux policy coreutils coreutils procps
3	Switch mode Switch permissive/enforcing mode	libselinux
4	User space AVC Use the access control feature of SELinux from userland applications	libselinux
5	Analyze policy Access data structure of policy	libsepol
6	Manage conditional policy Change parameters of conditional policy feature	libselinux libsepol libsemanage
7	Manage policy module framework Install and remove policy modules	libsemanage

tures. For example, data structures and APIs must be modified, which will also have an effect on userland. Therefore, we adopt another approach, in which we analyze overhead and bottlenecks first, and subsequently perform the tuning of the kernel.

3.1.2 Userland

SELinux userland was intended for server usage, therefore many features are unnecessary for embedded systems. File size can be reduced by selecting features that meet the following criteria.

- Access control feature of SELinux works.
- Security policy can be replaced.
  Most problems related to SELinux are caused by lack of policy configurations. To correct such problems, features that enable the user to replace security policy will be necessary.

Features in SELinux and userland can be classified as shown in **Table 4**. Features 1, 2, and 3 were selected according to the criteria above. Feature 1 is necessary to use access control, while 2 and 3 are required to replace policy.

In addition to removing unnecessary features, commands are integrated into BusyBox [18]. BusyBox is a tool that is widely used in embedded systems, and allows multiple commands to be executed from a single executable binary file. It reduces the overhead of executable file headers by integrating multiple commands into a single binary executable file. It is expected that merging SELinux-related commands to BusyBox will further reduce file size.

3.1.3 Security Policy

In PC systems, SELinux security policy is usually formulated by customizing a security policy template called refpolicy [19]. Customization is typically not required for PC server systems, because the necessary configurations have already been prepared. In contrast, substantial customization is required for embedded systems. For example, to reduce security policy size, unused configurations must be removed. In addition, because the file tree structure is different from PC systems, security policy must be modified to accommodate the structure. Such customization is dfficult, because there are more than 2,000 macros and 1,000 type configurations. Therefore, we prepared the security policy from scratch, without using refpolicy.

3.2 Modifications Acceptable to OSS Community

As discussed in Section 3.1, the source code of the Linux kernel, SELinux library, and BusyBox must be modified. To be acceptable to the related OSS community, these modifications must not adversely affect existing code. To avoid affecting existing code, the kernel, SELinux library, and BusyBox were modified as follows.

- Kernel
  The modifications do not alter the function of the kernel itself, because only tuning is required. However, there may be side effects that affect performance. For example, if memory usage is tuned, performance may decline. Therefore, we proposed modifications to the community that include an evaluation of possible regressions.
- SELinux library
  Our modifications remove features that are unnecessary for embedded implementations; however, these features are necessary for PC usage. To resolve this conflict, build flags in the Makefile and #ifndef preprocessor in the C language are used. The build flag **EMBEDDED** is defined in the Makefile, and unnecessary code is enclosed with #ifndef and #endif. When a build is executed with the **EMBEDDED** flag set to **y**, parameters are defined for #ifndef blocks and unnecessary source code is not compiled. Conversely, when a build is executed and the **EMBEDDED** flag is not set to **y**, this code is compiled.
- BusyBox
  BusyBox has a framework that switches included features on or off according to a build flag. The **CONFIG_SELINUX** build flag is already set to link the selinux library; we use this flag to include SELinux-related commands only when the **CONFIG_SELINUX** flag is enabled.

4. Implementation of Embedded SELinux

Embedded SELinux was implemented following the approaches discussed in the previous section.

4.1 Kernel Tuning

The Kernel was tuned to reduce read/write system call overhead and memory usage.

4.1.1 Reducing Read/write Overhead

Bottlenecks were analyzed by considering the read/write flow. A process behaves as follows when it reads from, or writes to, a file.

(1) Open file
  A process opens a file and obtains a file descriptor with the **open** system call, with a specified access mode (e.g., read only, write). Linux file permissions and SELinux permissions are checked.

(2) Read/write
  Using the file descriptor obtained from the **open** system call, a **read**/**write** system call is executed to complete the actual read and write operations. The access mode is checked first. If the access mode does not match the operation, the

read/write process returns a permission error. Next, SELinux permissions are checked. If the validation succeeds, data is read from, or written to, the opened file. Reads and writes are typically called from applications multiple times once a file is opened.

(3) Close
The file is closed by using the file descriptor, and related resources are released.

By examining the read/write flow, it is evident that the SELinux permission verification in the read/write step is duplicated, because an operation that is different from the one allowed in the open step is denied by the access mode check in the read/write step. For example, assume a process attempts to open a file in read-only mode; read access is allowed and write is not allowed in the SELinux policy. In this case, when the process attempts a write system call, access is denied in the access mode check because the file descriptor obtained in the open step only includes read-only access. Therefore, the SELinux permission check is not necessary in the read/write step. However, there are two exceptional cases where SELinux permission checks are necessary for a read/write step. First, if the security policy is changed between the time the file is opened and the time of the read/write call, permission must be rechecked for the read/write call to accommodate the change. Second, when the domain or type labels are changed in a similar manner, permission must be rechecked.

To perform SELinux permission checks only in the exceptional read and write cases described above, the following changes were introduced in the kernel.

(1) SELinux
A data structure was added to store the state of the security policy, as well as domain and type labels.

(2) Open system call
A new LSM hook function, **security_dentry_open**, was introduced. In **security_dentry_open**, the states of the security policy, domain, and type are stored in the data structure introduced in (1). SELinux file permissions are rechecked after the states are stored, to account for any state changes.

(3) Read/write system call
Compares the current status of the security policy, domain, and type with those described in (2). SELinux permission is checked only when a status change is detected.

4.1.2 Reducing Memory Usage

To reduce memory usage, unnecessary SELinux data structures were removed from the kernel. The largest data structure in SELinux is the hash table in **struct avtab**. Two avtabs are statically allocated in the Linux kernel, and security policy access rules are stored there. During this process, 32,768 hash slots are prepared for each hash table, which consumes approximately 250 Kbytes. In PC systems, the number of access rules often exceeds 100,000. Conversely, in embedded systems, the number of rules is often lower than 10,000, because embedded systems have significantly fewer installed applications compared to PC systems. Therefore, embedded systems require significantly fewer hash slots. To reduce the number of hash slots, they are allocated dynamically based on the number of access control rules in the security policy; i.e., the number of allocated hash slots is one-quarter the number of rules.

Table 5 SELinux commands ported to BusyBox.

#	Feature	Commands
1	Load policy	load_policy
2	Change labels	chcon,setfiles,restorecon -Z option for ls and ps
3	Switch mode	getenforce,setenforce,selinuxenabled

4.2 Modification of Userland Programs

4.2.1 Reducing Library Size

As described in Section 3.1.2, unnecessary features are disabled by the build flag, and only libselinux is used. The detailed modifications to libselinux are as follows.

- Remove libsepol function call
  Usage of libsepol, which is approximately 300 Kbytes in size, is forced because libselinux calls the libsepol function to load the security policy. To remove the dependency on libsepol, the security policy is loaded directly by calling kernel functions, thus bypassing libsepol.
- User space AVC and conditional policy are disabled
  When the **EMBEDDED** build flag is enabled, files related to these functions are not compiled, and code fragments related to them are disabled by the #ifndef preprocessor.

4.2.2 Integrating Commands to BusyBox

As discussed in Section 3.1.2, commands related to Table 4 1-3 are ported to BusyBox from the libselinux and policycoreutils packages using the **CONFIG_SELINUX** build flag. Ported commands are shown in **Table 5**. The -Z option logic to show labels when using ls and ps is embedded in ls and ps BusyBox applets. Other commands are also implemented as BusyBox applets. They are compiled only when the **CONFIG_SELINUX** flag is enabled.

4.3 Preparing Policy from Scratch

Writing SELinux security policy is difficult because there are hundreds of permissions and label configurations. To facilitate policy writing, we adopted SEEdit [20], which was proposed in our previous research [21]. SEEdit simplifies security policy configuration using a higher-level language called Simplified Policy Description Language (SPDL). SPDL reduces number of permissions by integrated permission, and hides label configuration by path name based configuration. Security policy is written using SPDL and converted into a kernel-loadable format by SEEdit.

5. Evaluation

Performance of embedded SELinux is measured on an evaluation board which is used for development of M2M gateway and CE devices and performance is compared with SELinux without tuning. Possible regression is also measured and the result of proposal to community is shown.

5.1 Target Device and Software

The devices and software versions used in the evaluation are listed below.

(1) Target device
We used a Renesas Technology R0P751RLC001RL

Table 6 Read/write system call execution time (unit: micro second) on the evaluation board, measured by lmbench and overhead of SELinux. The same security policy (files size is 60 Kbytes, 2,000 access rules are included) is used to eliminate the effects from differences in policy.

lmbench	SELinux disabled	Standard SELinux		Embedded SELinux	
Null read	2.39	5.49	(130.0%)	2.68	(12.5%)
Null write	2.07	5.10	(146.6%)	2.38	(14.9%)

Table 7 Read/write execution time (unit: micro second) on the evaluation board, measured by Unixbench and overhead of SELinux. The security policy is the same as Table 6.

Unixbench read/write	SELinux disabled	Standard SELinux		Embedded SELinux	
256 byte read	7.95	13.25	(66.6%)	9.24	(16.2%)
256 byte write	12.84	21.42	(66.8%)	16.29	(26.8%)
1,024 byte read	12.79	17.97	(40.5%)	14.46	(13.1%)
1,024 byte write	19.25	27.69	(43.9%)	22.89	(19.0%)
4,096 byte read	33.96	39.45	(16.2%)	35.08	(3.3%)
4,096 byte write	806.45	781.25	(−3.1%)	806.45	(0.0%)

(R2DPLUS) board as a target device. This board is often used to evaluate software for embedded devices such as M2M gateway and CE devices which are targets of embedded Linux. The specifications are shown below.

- CPU: SH7751R(SH4) 240 MHz
- RAM: 64 MBytes
- Compact flash: 512 MBytes
- Flash ROM: 64 MBytes (32 MBytes available for root file system)

SELinux can be ported to both compact flash and flash ROM. For convenience, we measured benchmarks using the compact flash system.

(2) Software and policy

The software versions and policy used in the evaluation are listed below.

- Kernel: Linux 2.6.22
- SELinux userland: libselinux 2.0.27
- Security policy (before tuning): policy.21 and file_contexts file were taken from selinux-policy-targeted-2.4.6-80.fc6, obtained from Fedora 6.
- Security policy (after tuning): Written by SELinux Policy Editor, including configurations for 10 applications. Not all applications are confined similar to targeted policy [22].

5.2 Benchmark Results

We ported standard SELinux and embedded SELinux to the target board, and measured benchmarks for both. The benchmark results for read and write overhead, file size, and memory usage are shown below.

5.2.1 System Call Overhead

Read and write system call overhead was measured by lmbench and unixbench. The results are listed in **Tables 6** and **7**. Before tuning, the SELinux overhead for read/write was significant. Null read/write overhead was reduced by 90% after tuning . The overhead in reading the 4,096 buffer was a significant problem; however, it was mostly eliminated in embedded SELinux. In addition, Yamamoto et al. [23] described the effectiveness of our modifications when testing them using a Pentium 4 PC.

Table 8 File size related to SELinux. Userlands are built with -Os flag and stripped.

Component	File size of standard SELinux (Kbyte)	File size of embedded SELinux (Kbyte)
Kernel (zImage) size increase	74	74
Library	482	66
Command	375	11 (39: without using BusyBox)
Policy file	1,356	60
Total	2,287	211

Table 9 Memory usage by SELinux.

Component	Memory usage of standard SELinux (Kbyte)	Memory usage of embedded SELinux (Kbyte)
Hash tables in struct avtab	252	1
SELinux program and policy	5,113	464
Total	5,365	465

5.2.2 File Size

The file sizes related to SELinux are summarized in **Table 8**. As a result of tuning, file sizes were reduced to 211 Kbytes from 2,287 Kbytes. On the evaluation board, the flash ROM available for the root file system is 32 MBytes. The size of SELinux is less than 1% of this capacity; therefore, it is acceptable for the evaluation board.

Integrating commands to BusyBox reduces overhead by 28 Kbytes. Commands require 4 Kbytes, and seven commands were ported. The effect will increase as the number of ported commands increases.

5.2.3 Memory Usage

We measured memory usage with the free command. The usage by SELinux was measured as follows.

A = The result of the free command when the SELinux-enabled kernel was booted;
B = The result of the free command when the kernel without SELinux was booted;
Therefore, memory usage by SELinux = A - B.

SELinux's memory usage was measured for both standard SELinux and embedded SELinux. We also measured the memory usage of the hash table in **struct avtab** to measure the effects of tuning. Code that displayed the size of the allocated tables was inserted into the kernel for that purpose. The results are shown in **Table 9**. The memory consumption after tuning was 465 Kbytes and the evaluation board's memory capacity is 64 MBytes. Therefore, SELinux consumes less than 1% of the board's memory capacity. This demonstrates that, the device has ample resources to accommodate SELinux.

5.3 Regressions

5.3.1 Possibility of Regressions

The modifications to the SELinux library and BusyBox do not affect existing features, because they are implemented as options. If developers want to use existing features as before, they only

Table 10 Open/close execution time (units in microseconds) on the evaluation board, measured by lmbench and overhead of SELinux. The security policy is the same as Table 6.

lmbench	SELinux disabled	Standard SELinux		Embedded SELinux	
Simple open/close	32.55	62.82	(93.0%)	58.70	(80.3%)

Table 11 Effect of reducing number of hash slot, when number of SELinux access rules is 8,188.

Number of hash slot	longest chain length	time to call security_compute_av (sec)
8,192 (=number of rules)	13	9.67
4,096 (=number of rules/2)	21	9.65
2,048 (=number of rules/4)	35	9.68
1,024 (=number of rules/8)	64	9.78

need to set the SELinux build flag to off. However, modifications to the kernel may affect the performance of existing features as described below.

- Tuning of read/write overhead
  This may affect performance when files are opened, because the new LSM hook function **security_dentry_open** was added.
- Tuning of struct avtab
  This may affect performance when the security policy is retrieved, because the hash table size was reduced.

5.3.2 Measurement of Side Effect

These possible side effects of our modifications were measured on the evaluation board as follows.

- Tuning of read/write overhead
  The performance of the file opening process was measured by lmbench for standard SELinux and for embedded SELinux.
- Tuning of struct avtab
  To measure performance of security policy retrievals, the processing time of **security_compute_av**, where security policy is searched from avtab, must be measured. The time required to call security compute av 10,000 times from the startup of SELinux was measured, because the time required to perform a single function call was too short to measure. The time was measured by varying the number of hash slots when the security policy (which contains 8,188 rules) was loaded.

5.3.3 Result

The results of side effect measurements are shown in **Tables 10** and **11**.

- Side effects on the performance of open
  The results of the performance measurements for the open system call are shown in Table 10. The performance does not decrease in embedded SELinux; it is actually better than standard SELinux. We did not determine the reason. One possible reason is that instructions generated by the C compiler were arranged more efficiently for embedded SELinux.
- Side effects on the performance of security policy searches
  The performance of the security policy search is shown in Table 11. No drop in performance was observed, even when the number of allocated hash slots was reduced to one quarter the number of access rules.

To summarize, no adverse side effects were caused by our modifications.

Table 12 OSS versions where our modifications are merged.

Modification	Merged OSS version
Reducing read/write overhead	Linux 2.6.24
Reducing size of avtab	Linux 2.6.24
Reducing size of libselinux	libselinux 2.0.35
Integrating SELinux commands	BusyBox 1.9.0

5.4 Results of OSS Community Proposals

We submitted our modifications to the Linux, BusyBox, and SELinux OSS communities, using the diff utility [24] to highlight the modifications; evaluation results were included as well. The modifications were successfully merged, as shown in **Table 12**. The links to the patches can be accessed on the Embedded Linux Wiki at http://elinux.org/SELinux.

6. Related Work

The initial research to port SELinux to embedded devices was performed by Coker [25]. SELinux was ported to an ARM-based mobile device. The version of SELinux used in this initial research had not been merged into the Linux kernel community code; thus, the features and data structures are different from the SELinux used in our work. The research focused on reducing memory and file size in userland commands and policy. His approach in reducing userland commands was similar to ours. Only necessary features were ported, and BusyBox was also used. However, the modifications were not merged to community code, and there is no mention of overhead or kernel memory tuning. Security policy was not created from scratch, because the security policy template for older versions of SELinux was simpler and easier to customize than refpolicy.

Another approach to reduce resource usage is to modify hardware. Fiorin et al. [26] reduced the overhead of SELinux for embedded devices by hardware assistance. They implemented an AVC on a FPGA, and succeeded in reducing SELinux overhead in system calls.

Our embedded SELinux provides the basic structure required to apply SELinux to embedded systems by reducing resource usage; however, there are two additional problems to be resolved. The first problem involves integration with application frameworks for embedded systems. Application frameworks have resources that are not visible to the OS, and they control access to such resources by various permissions in userland. Integrating SELinux to application frameworks will bring MAC to the application layer. Research that aims to resolve these issues has been performed for BusyBox and Android. BusyBox is a simple application framework in which there is a problem assigning domains to each command, i.e., commands in BusyBox are not identified by the OS, and all commands executed from BusyBox are assigned the same domain. This problem was resolved by Shinji [27], by utilizing APIs provided by SELinux and the solu-

tion was subsequently merged into BusyBox's community code.

Android is a widely used application framework for mobile devices, and the OS is based on the Linux kernel. Android provides its own application layer permission check, and these applications are not identified by the OS. The initial port of SELinux to Android [29] did not resolve the problem. SEAndroid [13] brings the SELinux check to Android applications by inserting SELinux APIs into Android's framework. However, Android can only accommodate mobile devices with GUIs. There are frameworks for devices without GUIs, such as OSGi [30] and Alljoyn [31]. Integrating SELinux to these frameworks will be required in the future.

The second problem involves developing and managing security policies. SEEdit is used in our work to facilitate security policy development; however, it does not provide a mechanism to manage the security policy. Honda et al. [28] proposed an SELinux policy configuration system for mobile devices. This system enables modular management of security policies.

7. Conclusion

Applying SELinux to embedded devices such as M2M gateways and CE devices where CPU is around 200 Mhz to 600 Mhz and RAM size is often around 64 MBytes, presents several challenges. The SELinux kernel, userland, and the security policy all consume an unacceptable amount of hardware resources. Related code must be tuned, and tuned codes must be merged into OSS community code to benefit from usage on the its ecosystem.

We developed a version of embedded SELinux in which resource usage is reduced, using code modifications that are acceptable to the OSS community. Resource usage is reduced using three techniques. First, the Linux kernel is tuned to reduce CPU overhead and memory usage. Second, unnecessary code is removed from userland libraries and commands. Third, security policy size is reduced by our policy-writing tool. To be acceptable to the OSS community, the side effects of the code modifications were measured. In addition, the proposed system allows removed code to be selected by using a build flag at compilation time.

Embedded SELinux was evaluated on a SH-based evaluation board targeted for M2M gateways and CE devices. Benchmark results show that the SELinux overhead for read and write operations is almost negligible. SELinux's file space requirements are approximately 200 Kbytes, and memory usage is approximately 500 Kbytes, which consume approximately 1% of the flash ROM and RAM of the evaluation board, respectively. These results show that SELinux can be applied to embedded devices effectively. Regressions were also measured, and no regression problems were observed. We presented the modified code to the OSS community along with the evaluation results; as a result, the modifications were successfully merged into the community code.

Acknowledgments We are helped by many people during the work. We would like to thank them. We ported SELinux commands to BusyBox with people in sebusybox project. KaiGai Kohei developed chcon applet, Hiroshi Shinji developed setenforce and getenforce applets. They also reviewed applets and gave us useful comments. People in SELinux community reviewed modification of kernel and library, and gave us feedback. People in BusyBox community reviewed proposed patches to BusyBox and gave us feedback.

References

[1] Wu, G., Talwar, S., Johnsson, K., Himayat, N. and Johnson, K.: M2M: From mobile to embedded internet, *IEEE Communication Magazine*, Vol.49, No.4, pp.36–43 (2011).
[2] ETSI TS 102 690 V1.1.1: Machine-to-Machine communications (M2M) functional architecture (2011).
[3] Bojinov, H., Boneh, D., Cannings, R. and Andmalchev, I.: Address space randomization for mobile devices, *Proc. 4th ACM Conference on Wireless Network Security*, pp.127–138 (2011).
[4] Schwartz, E., Avgerinos, T. and Brumley, D.: Q: Exploit Hardening Made Easy, *Proc. 20th USENIX Security Symposium*, pp.379–394 (2011).
[5] Strackx, R., Younan, Y., Philippaerts, P., Piessens, F., Lachmund., S. and Walter, T.: Breaking the memory secrecy assumption, *Proc. 2nd European Workshop on System Security*, pp.1–8 (2009).
[6] Loscocco, P., Smalley, S., Muckelbauer, P., Taylor, R., Turner, S. and Farrell, J.: The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments, *Proc. 21st National Information Systems Security Conference*, pp.303–314 (1998).
[7] Wright, C., Cowan, C., Smalley, S., Morris, J. and Kroah-Hartman, G.: Linux Security Modules: General Security Support for the Linux Kernel, *Proc. 11th USENIX Security Symposium*, pp.17–31 (2002).
[8] Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System, *Proc. FREENIX Track of the 2001 USENIX Annual Technical Conference*, pp.29–42 (2001).
[9] Harada, T., Handa, T., Hashimoto, M. and Tanaka, H.: Mandatory Access Control Method Based on Application Execution State, *IPSJ Journal*, Vol.53, No.9, pp.1–18 (2012). (in Japanese)
[10] Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P. and Gligor, V.: SubDomain: Parsimonious Server Security, *Proc. 14th USENIX Conference on System Administration*, pp.355–368 (2000).
[11] Boebert, W. and Kain, R.: A Practical Alternative to Hierarchical Integrity Policies, *Proc. 8th National Computer Security Conference*, pp.18–27 (1985).
[12] Saini, M. and Kaur, K.: Review of Open Source Software Development Life Cycle Models, *International Journal of Software Engineering and Its Applications*, Vol.8, No.3, pp.417–434 (2014).
[13] Smalley, S. and Craig, R.: Security Enhanced (SE) Android: Bringing Flexible MAC to Android, *Proc. 20th Network and Distributed System Security symposium*, pp.20–38 (2013).
[14] Petazzoni, T. and Shotwell, C.: busybox: Add option to enable SELinux support, Buildroot mailing list (2013), available from ⟨http://lists.busybox.net/pipermail/buildroot/2013-December/085067.html⟩.
[15] Spencer, R., Smalley, S., Hibler, M., Andersen, D. and Lepreau, J.: The Flask Security Architecture: System Support for Diverse Security Policies, *Proc. 8th USENIX Security Symposium*, pp.123–139 (1999).
[16] McVoy, L. and Staelin, C.: lmbench: Portable tools for performance analysis, *Proc. USENIX 1996 Annual Technical Conference*, pp.279–295 (1996).
[17] UNIX Bench, available from ⟨https://code.google.com/p/byte-unixbench/⟩.
[18] BusyBox, available from ⟨http://www.busybox.net/⟩.
[19] Refpolicy, available from ⟨http://oss.tresys.com/projects/refpolicy⟩.
[20] SELinux Policy Editor, available from ⟨http://seedit.sourceforge.net/⟩.
[21] Nakamura, Y., Sameshima, Y. and Yamauchi, T.: SELinux Security Policy Configuration System with Higher Level Language, *Journal of Information Processing*, Vol.18, pp.201–212 (2010).
[22] Coker, F. and Coker, R.: Taking advantage of SELinux in Red Hat® Enterprise Linux®, *Red Hat Magazine* (2005), available from ⟨http://www.redhat.com/magazine/006apr05/features/selinux/⟩.
[23] Yamamoto, K. and Yamauchi, T.: Evaluation of Performance of Secure OS Using Performance Evaluation Mechanism of LSMPMON, *IPSJ Journal*, Vol.52, No.9, pp.2596–2601 (2011). (in Japanese)
[24] GNU Diffutils, available from ⟨http://www.gnu.org/software/diffutils/⟩.
[25] Coker, R.: Porting NSA Security Enhanced Linux to Hand-held devices, *Proc. 2003 Ottawa Linux Symposium*, pp.117–127 (2003).
[26] Fiorin, L., Ferrante, A., Padarnitsas, K. and Carucci, S.: Hardware-assisted security enhanced Linux in embedded systems, *Proc. 5th Workshop on Embedded Systems Security*, article No.3 (2010).
[27] Shinji, H.: Domain assignment support for SELinux/AppArmor/LIDS, BusyBox mailing list, available from ⟨http://www.busybox.net/lists/busybox/2007-August/028481.html⟩.
[28] Honda, A., Asakura, Y., Saida, Y. and Watanabe, M.: Policy addition mechanism of secure OS for embedded system, *IPSJ SIG Techinical*

Report, Vol.2008, No.21, pp.109–114 (2008). (in Japanese)
[29] Shabtai, A., Fledel, Y. and Elovici, Y.: Securing Android-Powered Mobile Devices Using SELinux, *IEEE Security and Privacy Magazine*, pp.36–44 (2010).
[30] OSGi Alliance, available from ⟨http://www.osgi.org⟩.
[31] Alljoyn, available from ⟨https://www.alljoyn.org/⟩.

Yuichi Nakamura received his B.S. and M.S. degrees in physics from the University of Tokyo in 1999 and 2001, and M.S. degree in computer science from The George Washington University in 2006. He worked for Hitachi Solutions from 2001 to 2015. He is working for Hitachi since 2015 and is also studying at Okayama University to obtain a Ph.D. degree. He is a member of IPSJ.

Yoshiki Sameshima received his B.S. degree from Kyoto University in 1984, M.S. degree in Mathematics from Osaka University in 1986, and Ph.D. degree in computer science and technology from Osaka University in 2008. He has been working for Hitachi Solutions since 1986, and was at Computer Science Department of University College London from 1992 to 1994. Since 1993, he has been doing research on information security. He is a member of IEICE, IPSJ, JSSM, the USENIX Association.

Toshihiro Yamauchi received his 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, USENIX and IEEE.