Reprinted from the

Proceedings of the Linux Symposium

July 23rd–26th, 2008
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
Matthew Wilson, *rPath*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
Eugene Teo, *Red Hat, Inc.*
Kyle McMartin, *Red Hat, Inc.*
Jake Edge, *LWN.net*
Robyn Bergeron
Dave Boutcher, *IBM*
Mats Wichmann, *Intel*

SELinux for Consumer Electronics Devices

Yuichi Nakamura
Hitachi Software Engineering
ynakam@hitachisoft.jp

Yoshiki Sameshima
Hitachi Software Engineering
same@hitachisoft.jp

Abstract

As the number of network-connect Consumer Electronics (CE) devices has increased, the security of these devices has become important. SELinux is widely used for PC servers to prevent attacks from a network. However, there are problems in applying SELinux to CE devices. SELinux kernel, userland, and policy consume hardware resources unacceptably. This paper describes tuning SELinux for use in CE devices. The tuning has two features. The first is using our policy writing tool to reduce the policy size. It facilitates writing small policy by simplified policy syntax. The second is tuning the SELinux kernel and userland. We have tuned permission check, removed needless features for CE devices, and integrated userland to BusyBox. We have evaluated tuned SELinux on a SuperH (SH) processor based device, and found the consumption of hardware resources to be acceptable for CE devices.

1 Introduction

Linux is a leading OS for embedded systems [1], and has also been adopted in CE devices such as TVs, DVD recorders, set top boxes, mobile phones, and home gateways. Because CE devices are connected to the Internet, their security is now an important issue. Once vulnerabilities in CE devices are exploited by attackers, they can destroy the system, steal information, and attack others.

1.1 Requirements of security technologies for CE devices

To counter security problems, security technologies are necessary. However, security technologies for CE devices have to meet the following three requirements.

1. Effective without update
   The security technlogies have to be effective even without security updates, because the process of updating introduces several problems. Some CE devices do not have a network updater. To fix vulnerabilities for such devices, manufacturers have to recall devices, and re-write flash ROM. Even if devices do have a network updater, security patches tend to be delayed or not provided, because preparing updates is a heavy task for manufacturers. Updates for CE devices are provided from the manufacturers, not from OS distributors. The manufacturers have to track all vulnerabilities and develop security patches as soon as possible; as a result, manufacturers will likely give up providing updates.

2. Architecture-independent
   The security technologies have to be architecture-independent, since many CPU architecures—such as SuperH (SH), ARM, MIPS, PowerPC, and x86—are used in CE devices. Moreover, there are many variants within a CPU family. For example, SH has SH2, SH3, SH4, and SH64 variants. To port security technologies for such various CPUs, the security technologies need to be architecture-independent.

3. Small resource usage
   The security technologies have to work in resource-constrained environments. The architecture of a CPU is focused on power consumption rather than speed, thus the CPU clock is often slow such as 200Mhz. Main memory is often less than 64Mbyte, and the file storage area is often less than 32Mbyte to reduce hardware cost.

1.2 Porting security technologies to CE devices

Many security technlogies are already used in the PC environment. The most major ones are: buffer overrun protection, network updater, and anti-virus software.

However, they do not meet the previously stated requirements for CE devices. First, buffer overrun protections are architecture-dependent, because they depend on memory management of the CPU. For example, Exec Shield [3] is one of the most widely used buffer overrun protection technologies. It modifies codes under the `arch` directory in the kernel source tree. Second, network updaters such as yum [2] obviously do not meet the first requirement of being effective without update. Finally, anti-virus software does not meet the third requirement, because the pattern file consumes file storage area, sometimes more than 30Mbyte in a PC environment. In addition, the system becomes slow when a virus scan is running.

1.3 SELinux for CE devices

The Linux kernel includes Security-Enhanced Linux (SELinux) [4]. SELinux meets the first requirement, because it is effective even when a security update is not applied. SELinux provides label-based access control to Linux. Each process has a label called domain, and each resource has a label called type. Access rules between the domains and types are described in the security policy. Domains are configured to access only types that they need; thus, processes have only limited access rights. Assuming that a vulnerability exists in an application and it is not fixed, attackers can take control of the application. However, attack attempts usually fail due to lack of access rights; attackers obtain the domain of the application process that is allowed to access only limited types. SELinux has actually been widely used for PC servers and blocked attacks [5]. SELinux also meets the second requirement of being architecture-independent; there is no code modification under `arch` directory. However, SELinux does not meet the third requirement—small resource usage. SELinux was focused on PC usage, and many features were added. Consequently, its resource consumption has become unacceptable for CE devices.

The purpose of our work is to apply SELinux to CE devices. SELinux is tuned to meet resource requirements for that purpose. Our tuning has two features. The first is reducing policy size by using our policy writing tool. The tool facilitates writing small policy by simplifiying policy syntax. The second is tuning the SELinux kernel and userland. Permission checks are tuned, needless features for CE devices are removed, and userland commands are integrated to BusyBox [6]. Tuned SELinux is evaluted on a SH-based device. SH is a CPU family widely used for CE devices, including DVD recorders and home gateways. The evaluation results show the consumption of hardware resources is acceptable for CE devices.

2 Problems in applying SELinux to CE devices

To apply SELinux to CE devices, SELinux has to meet resource usage requirements. However, SELinux consumes CPU, file size, and memory unacceptably when used in CE devices. The detail is described in this section.

2.1 CPU usage

SELinux has overhead for system calls (syscalls) because of its security checks. In the PC environment, P. Loscocco et al. [4] measured the overhead and concluded that it is insignificant. However, the overhead is a problem when using SELinux on a CE device platform. The SELinux overhead measured on a CE device platform is shown in Table 1 and Table 2. We measured them by lmbench [7] and Unixbench [8]. Values in the tables are an average of 5 trials. The CE device platform is SH7751R (SH4 architecture, 240Mhz) processor, Linux 2.6.22. In particular, the read/write overhead is a problem, because they are executed frequently and the overhead is big. Overhead more than 100% is observed in null read/write; it is about 10% when measured in a Pentium 4 PC. Moreover, 16% overhead remains in reading a 4096-byte buffer. 4096 bytes are often used for I/O buffer because it is the page size in many CPUs for embedded systems, such as SH and ARM.

2.2 File size increase

The file size of the kernel and userland increases when SELinux is ported because of components listed in Table 3. The increase is about 2Mbyte if SELinux for PC (SELinux included in Fedora Core 6) is ported without tuning. However, the increase is not acceptable for CE devices, because flash ROM less than 32Mbyte is often used to store a file system. If SELinux consumes 2Mbyte, it is too much.

lmbench	Overhead(%)
Null read	130
Null write	147
Stat	97
Create	163
Unlink	86
Open/close	93
Pipe	67
UNIX socket	31
TCP	22
UDP	28

Table 1: System call overhead by SELinux on a CE device platform, measured by lmbench. To compute overhead, the time to execute syscall in SELinux disabled kernel is used as the baseline value.

Unixbench read/write	Overhead(%)
256 byte read	66.6
256 byte write	66.8
1024 byte read	40.5
1024 byte write	43.9
4096 byte read	16.2
4096 byte write	-3.1

Table 2: The SELinux overhead for read/write on a CE device platform, measured by Unixbench. To compute overhead, the throughput in SELinux disabled kernel is used as the baseline value.

2.3 Memory consumption

SELinux has data structures in the kernel to load the security policy. The memory consumption by the security policy used in PC is about 5Mbyte. However, it is also unacceptable for CE devices because the size of RAM is often less than 64Mbyte and swap is not prepared. If SELinux is used for CE devices, the possibility that memory can not be allocated increases. If memory can not be allocated, applications will not work correctly.

3 Tuning SELinux for CE devices

Tuning is needed because the resource consumption by SELinux is not acceptable for CE devices, as described above. Our tuning consists of two parts. The first is reducing policy size by utilizing our policy writing tool. The second part is tuning the kernel and userland.

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

Table 3: Files related to SELinux

3.1 Reducing policy size by policy writing tool

Since the security policy consumes both file storage area and RAM, the security policy has to be small. Problems in preparing a small policy and our approach to resolving them are described.

3.1.1 Problems in preparing small policy

To prepare policy, refpolicy [9] is usually used and customized for the target system [10]. Refpolicy is a policy developed by the SELinux community, and is used in distributions such as Red Hat and Fedora by default. It is composed of sample configurations for many applications and a set of macros. Refpolicy works well on PC systems, but it is hard to use for CE devices. To prepare small policy based on refpolicy, one has to remove unnecessary configurations, then add necessary ones. However, there are three difficulties in the process.

1. Large amount of removal
   The amount of removal is large, because configurations for many distributions, applications, and use cases are included. For example, configurations for many Apache modules are included in refpolicy. To configure Apache that serves simple home page and CGI, configurations about unnecessary Apache modules have to be removed. We removed about 400 lines for that. For each application, such removal has to be done.

2. Many macros and labels
   Refpolicy contains many macros and labels. More

than 2,000 macros are defined and used. More than 1,000 labels are declared. Policy developers have to understand them in removing and adding configurations. Figure 1 is an example of configurations in refpolicy. It is hard for policy developers to understand so many macros and labels.

```
policy_module(apache,1.3.16)
type httpd_t;
type httpd_exec_t;
init_daemon_domain(httpd_t,httpd_exec_t)
role system_r types httpd_t;
....
ifdef(`targeted_policy',`
typealias httpd_sys_content_t
        alias httpd_user_content_t;
typealias httpd_sys_script_exec_t
        alias httpd_user_script_exec_t;
')
allow httpd_t httpd_sys_content_t:
dir r_dir_perms;
...
corenet_tcp_sendrecv_all_if(httpd_t)
...
```

Figure 1: Example of configurations used in PC. This is part of configuration of http server.

3. Dependency
   Two kinds of dependencies that appear in removing and adding configurations increase the cost of preparing policy.

   (a) Labels and declarations
   There are dependencies in labels (domain/type) and declaration. They make removing configurations difficult. The major part of SELinux configuration is allowing domain to access some type, like below.
   `allow httpd_t sendmail_exec_t: file execute;`
   This is part of configuration to allow web server to send mails. In SELinux policy syntax, all labels must be declared like below.
   `type httpd_t;`
   `type sendmail_exec_t;`
   Such text based policy configuratin is converted to binary representation to be loaded in the kernel. If the declaration is removed, error is outputted and conversion fails. This often happens in removing files. Refpolicy is composed of many files. For example, in file mta.te configurations related to sendmail is described and sendmail_exec_t is declared. If mta.te is removed, policy conversion fails in `allow httpd_t sendmail_exec_t: file execute;`. This line has to be removed to convert policy successfully.

   (b) Labeling change
   Dependencies also appear when labeling is changed. Assume an application `foo` running as `foo_t` domain is allowed to access `foo_file_t` type and under `/foo` directory are labeled as `foo_file_t`. Then `foo` can access under `/foo`. What happens an application `bar` running as `bar_t` domain needs configuration to access `/foo/bar`? One will define new type such as `bar_file_t` and labels `/foo/bar` as `bar_file_t` type, then allow `bar_t` to access `bar_file_t`. Problem is happenning here. `foo` can not access `/foo/bar`, because `foo_t` is not allowed to access `bar_t`. To resolve that, configuration that allows `foo_t` to access `bar_t` has to be described. In adding new configuration, such dependency have to be considered carefully.

3.1.2 Preparing policy by SELinux Policy Editor

Policy is prepared without using refpolicy to avoid above difficulties. SELinux Policy Editor (SEEdit) [11] is used for that. SEEdit was developed by the authors to facilitate policy writing. The main feature is Simplified Policy Description Language (SPDL). Fig 2 is an example of configuration written by SPDL. Type labels are hidden; in other words, file names and port numbers can be used to specify resources. SPDL is converted to usual SELinux policy expression by converter, and policy is applied.

```
domain httpd_t;
program /usr/sbin/httpd;
allow /var/www/** r;
allownet -protocol tcp -port 80 server;
```

Figure 2: Example of SPDL, part of configuration for http server

How difficulties described in Section 3.1.1 are resolved by SEEdit is shown below.

1. Large amount of removal
   Only configurations that are necessary for CE devices are described by SEEdit. Obviously, there is no need to remove unnecessary configurations. One has to create necessary configurations, but the number of lines to be described is small. For example, we wrote about 20 lines for web server that serves a simple homepage.

2. Macros and labels
   Labels are hidden, and macros are not used in SPDL. Syntax of SPDL appears instead of macros, but it is much simpler than macros.

3. Dependencies
   In SPDL, such dependencies are not included. Dependencies are resolved internally when SPDL is converted to the original SELinux configuration syntax.

3.2 Tuning the kernel and userland

SELinux was developed for PC usage, so there are unneeded features, functions, and data structures for CE devices. The SELinux overhead for syscalls, file size, and memory usage can be reduced by removing them. The removal strategy and implementation are described in this section.

3.2.1 Reducing overhead

The SELinux overhead is reduced by removing unneeded functions and redundant permission checks from the kernel.

1. Removal of function calls from SELinux access decision code
   Function calls can be removed from SELinux access decision function (`avc_has_perm`) by using inline functions and calling `avc_audit` only when it is necessary. `avc_has_perm` is called in all permission checks, thus the removal will reduce overhead.

2. Removal of duplicated permission checks in file open and read/write
   There are duplicated permission checks in the process of file open and read/write to the file descriptor. For example, when a process opens a file to read or write, read/write permission is checked. Read/write permission is checked again in every read/write system call to the file descriptor. The check at read/write time is duplicated, because read/write permission is already checked at open time. Therefore, the permission check at read/write time can be removed. There is one exception: When security policy is changed between file open and read/write time, permission has to be checked at read/write time to reflect the change.

3. Removal of permission checks related to network
   In the process of network communication, SELinux permissions are checked for NIC, IP address, and port number. Permission checks in NIC and IP address are removed, because they are rarely used. Note that if there is a domain that wants to communicate with only a specific IP address, they can not be removed.

3.2.2 Reducing file size

SELinux userland was intended for server usage, so many features are unnecessary for CE devices. File size can be reduced by choosing features that meet the following criteria.

- Access control feature of SELinux works.

- Security policy can be replaced.
  Most of the troubles related to SELinux are caused by a lack of policy configurations. To fix these issues, we need a feature to replace policy.

Features in SELinux userland can be classified like Table 4. Features 1 , 2, and 3 are chosen according to criteria above. Feature 1 is necessary to use access control; Features 2 and 3 are needed to replace policy.

Features 1 through 3 are chosen and implemented. In the implementation, commands are integrated to BusyBox, and libselinux is modified. By using BusyBox, the size can be reduced more. Libselinux was also tuned.

#	Feature	Description	Related packages
1	Load policy	Load security policy file to the kernel	libselinux
2	Change labels	View and change domains and types	libselinux policycoreutils 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

Table 4: Features included in SELinux userland. We use 1,2, and 3 for CE devices.

Unnecessary features were removed and the dependency on libsepol, whose size is about 300Kbytes, was removed.

3.2.3 Reducing memory usage

To reduce memory usage, unnecessary data structures are removed from the kernel. The biggest data structure in SELinux is the hash table in `struct avtab`. Two avtabs are used in the kernel. Access control rules in the security policy are stored in hash tables of `struct avtab`. 32,768 hash slots are prepared for each hash table; they consume about 260Kbyte. Hash slots were shrunk to reduce the size of avtabs. In addition, hash slots are allocated dynamically based on number of access control rules in policy, i.e. the number of allocated hash slots is 1/4 of the number of rules. That change creates a concern about performance, because the hash chain length will increase. However, although the chain length becomes longer, regressions in performance were not observed.

4 Evaluation

To evaluate our tuning, we ported SELinux to an evaluation board and measured performance both before and after tuning.

4.1 Target device and software

The specification of the device and version of the software used in the evaluation are shown.

1. Target device
   The Renesas Technology R0P751RLC001RL (R2DPLUS) board was used as our target device. This board is often used to evaluate software for CE devices. The specification is shown below.

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

   SELinux can be ported to both compact flash and flash ROM. We measured the benchmark on a compact flash system for convenience.

2. Software and policy
   The version of the software and policy used in the evaluation are listed below.

   - Kernel: Linux 2.6.22
   - SELinux userland: Obtained from SELinux svn tree (selinux.svn.sourceforge.net) as of Aug 1, 2007

- Security policy (before tuning): policy.21 and file_contexts file were taken from selinux-policy-targeted-2.4.6-80.fc6 (included in 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 [12].

4.2 Benchmark results

We ported SELinux to the target board and measured the benchmark before and after tuning. The benchmark results for syscall overhead, file size, and memory usage are shown.

4.2.1 Syscall overhead

Syscall overhead was measured by lmbench and unixbench. The result is shown in Table 5 and Table 6. The SELinux overhead for read/write was significant before tuning. Null read/write overhead is reduced to 1/10 of the previous overhead. The overhead in reading a 4096 buffer was especially problematic, but it is almost eliminated by our tuning.

lmbench	Overhead before tuning(%)	Overhead after tuning(%)
Null read	130	13
Null write	147	15
Stat	97	59
Create	163	146
Unlink	86	70
Open/close	93	62
Pipe	67	31
UNIX socket	31	6
TCP	22	11
UDP	28	12

Table 5: The SELinux overhead for system call on the evaluation board, measured by lmbench. Average of 5 trials.

4.2.2 File size

The file size related to SELinux is summarized in Table 7. As a result of tuning, the file size increase is reduced to 211Kbyte. In the evaluation board, flash ROM available for root file system is 32Mbyte. The size of SELinux is less than 1%, so it is acceptable for the evaluation board.

Unixbench read/write	Overhead before tuning(%)	Overhead after tuning(%)
256 byte read	66.6	16.2
256 byte write	66.8	26.8
1024 byte read	40.5	13.1
1024 byte write	43.9	19.0
4096 byte read	16.2	3.3
4096 byte write	-3.1	0

Table 6: The SELinux overhead for read/write on the evaluation board, measured by Unixbench. Average of 5 trials.

Component	File size before tuning(Kbyte)	File size after tuning(Kbyte)
Kernel(zimage) size increase	74	74
Library	482	66
Command	375	11
Policy file	1,356	60
Total	2,287	211

Table 7: File size related to SELinux. Userlands are built with -Os flag and stripped.

4.2.3 Memory usage

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

A = The result of `free` when SELinux enabled kernel booted.
B = The result of `free` command when SELinux disabled kernel booted.
Memory usage by SELinux = A - B

The memory usage by SELinux was measured for both before tuning and after tuning. We also measured memory usage of a hash table in `struct avtab` to see the effect of tuning. We inserted code that shows the

size of allocated tables for that purpose. The result is shown in Table 8. The memory consumption after tuning is 465Kbyte. In the evaluation board, memory size is 64Mbyte. The consumption by SELinux is less than 1%—small enough.

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

Table 8: Memory usage by SELinux

5 Related works

R. Coker [13] ported SELinux to an ARM-based device, but SELinux was Linux 2.4 based. Since then, SELinux has changed a lot. Implementation has changed, many features were added, and policy development process has changed. KaiGai [14] ported xattr support to jffs2 and was merged to Linux 2.6.18. We are using his work when we run SELinux on a flash ROM system. The seBusyBox project in the Japan SELinux Users Group worked to port SELinux commands and options to BusyBox. We joined this project and did the porting together. Applets ported in this project were merged to BusyBox. In this project H. Shinji [15] also worked to assign different domains to applets, and his work was merged to BusyBox 1.8.0. H. Nahari [16] presented a design of a secure embedded system. He also mentioned SELinux for embedded devices, but the detail was not described.

6 Conclusion

There are problems in applying SELinux to CE devices. SELinux kernel, userland, and the security policy consume hardware resources unacceptably. We tuned SELinux to meet the resource requirements of CE devices. The tuning has two features. The first is using our policy writing tool to reduce policy size. It facilitates writing small policy by simplifying policy syntax. The second is tuning SELinux kernel and userland. Permission checking was tuned, needless features for CE devices were removed, and userland was integrated to BusyBox. Tuned SELinux was evaluated on a SH-based CE device evaluation board. The benchmark result shows that the SELinux overhead for read/write is almost negligible. File size is about 200 Kbyte, and memory usage is about 500Kbyte, about 1% of the flash ROM and RAM of the evaluation board. We conclude that SELinux can be applied to CE devices easier as the result of our work.

7 Future works

There are remaining issues to be done in the future.

1. Xattr for file systems on flash ROMs
   There are several files systems for flash ROMs, including jffs2, yaffs2, and logfs. Jffs2 supports xattr; yaffs2 and logfs do not support xattr. The porting of xattr is needed to use SELinux on such file systems.

2. Strict policy
   We adopted targeted policy in this paper. We have to write strict policy for more security. That is to say, domains for every program are prepared. The number of access control rules and policy size will be large. To write small strict policy is a remaining problem.

8 Availability

We have submitted related patches to Linux, BusyBox and SELinux community. The merged works are shown in Table 9. The links to patches can be seen on the Embedded Linux Wiki, `http://elinux.org/SELinux`. The SELinux Policy Editor is available on the SELinux Policy Editor Website [11]. 2.2.0 supports the writing of policy for CE devices.

Work	Merged 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

Table 9: Availability of our work

9 Acknowledgements

We are helped by many people during the work. We would like to thank them. seBusyBox project community members and KaiGai Kohei (NEC) gave us useful comments. People in the SELinux community gave us feedback, Stephen Smalley gave us ideas for the implementation of tuning. People in the BusyBox community, especially Denys Vlasenko, reviewed patches and gave us feedback. Yusuke Goda (Renesas Solutions) provided a driver for flash ROM.

References

[1] Linux Devices.com: Linux to remain a leading embedded OS, says analyst (2007), http://www.linuxdevices.com/news/NS2335393489.html

[2] Yum: Yellow dog Update, Modified: http://linux.duke.edu/projects/yum/

[3] A. van de Ven: Limiting buffer overflows with ExecShield, Red Hat Magazine, July 2005 (2005), http://www.redhat.com/magazine/009jul05/features/execshield/

[4] P. Loscocco and S. Smalley: Integrating Flexible Support for Security Policies into the Linux Operating System: the Proceedings of the FREENIX Track of the 2001 USENIX Annual Technical Conference (2001)

[5] D. Marti: A seatbelt for server software: SELinux blocks real-world exploits, Linuxworld.com (2008), http://www.linuxworld.com/news/2008/022408-selinux.html

[6] BusyBox, http://www.busybox.net/

[7] L. McVoy and C. Staelin: lmbench: Portable tools for performance analysis: the Proceedings of USENIX 1996 Annual Technical Conference (1996)

[8] UNIX Bench, http://www.tux.org/pub/tux/benchmarks/System/unixbench/

[9] Refpolicy, http://oss.tresys.com/projects/refpolicy

[10] F. Mayer, K. MacMillan and D. Caplan: SELinux by Example, Prentice Hall (2006)

[11] SELinux Policy Editor, http://seedit.sourceforge.net/

[12] F. Coker and R. Coker: Taking advantage of SELinux in Red Hat® Enterprise Linux®, Red Hat Magazine, April 2005 (2005), http://www.redhat.com/magazine/006apr05/features/selinux/

[13] R. Coker: Porting NSA Security Enhanced Linux to Hand-held devices, Proceedings of the Linux Symposium, Ottawa, Ontario, Canada (2003)

[14] KaiGai: Migration of XATTR on JFFS2 and SELinux, CELF Jamboree 11(2006), http://tree.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree11

[15] H. Shinji: Domain assignment support for SELinux/AppArmor/LIDS, BusyBox mailing list, http://www.busybox.net/lists/busybox/2007-August/028481.html

[16] H. Narari: Trusted Secure Embedded Linux: From Hardware Root of Trust to Mandatory Access Control, Proceedings of the Linux Symposium, Ottawa, Ontario, Canada (2007)