Extending Linux for Multi-Level Security

George Wilson
IBM Corporation

Klaus Weidner
atsec information security corporation

Loulwa Salem
IBM Corporation

Abstract

Linux™ distributions have received numerous Common Criteria certifications in the last few years. Building on the recent Controlled Access Protection Profile (CAPP) certifications, an Open Source development effort to make Linux compliant with the Labeled Security Protection Profile (LSPP) and Role-Based Access Control Protection Profile (RBACPP) has been ongoing for almost two years. Development included adding and augmenting features of SELinux and other Linux components. This paper explores the evolution of, and rationale behind, the features developed to meet LSPP and RBACPP, and it discusses the current state of development and lessons learned.

1 Introduction

Common Criteria certification of Linux distributions has proceeded incrementally since the first certification was completed in 2003. Linux has made considerable progress in terms of meeting additional functional requirements at increasing assurance levels. Most recently, a community of software developers from a variety of organizations has been working to add the features required for certification against LSPP [20] and RBACPP [21]. The foremost LSPP functional requirement is that the target operating system must enforce a mandatory Multilevel Access Control (MLS) security policy, implemented with subject and object labels and enforcing Bell-LaPadula (BLP) [8] access rules on operations.

SELinux provides a flexible framework for defining security rules. However, it is primarily focused on a Type Enforcement (TE) policy. SELinux support for an MLS policy has been available for a while, but it has not been easily usable due to missing tool support. Fedora and Red Hat Enterprise Linux (RHEL) will soon have the infrastructure necessary to set up a system meeting LSPP requirements.

2 Linux Common Criteria Certification History

Linux Common Criteria certification activities have evolved from the basic documentation and test of preexisting functionality that comprised the earliest certification to the development of numerous new security features required to meet the MLS requirements for the LSPP/RBACPP certification. Each certification after the first has been based upon its predecessors. An overview of Linux Common Criteria certification history helps explain the present LSPP/RBACPP certification effort's origins.

2.1 Initial EAL2+ Certification

SUSE Linux Enterprise Server (SLES) 8 received the first certification in August 2003 at Evaluation Assurance Level 2 augmented with a flaw remediation process (EAL2+) against a limited security target [14]. Up to that time, some had questioned the feasibility of certifying an Open Source operating system under the Common Criteria [22].

The Common Criteria standard allows certification against a Security Target that is not based on a protection profile. SLES 8 provided basic Discretionary Access Control (DAC) capabilities, along with Identification and Authentication (I&A). These capabilities were sufficient to write a Security Target substantial enough to certify it at an EAL2+ level based on a subset of CAPP [5]. The Security Target is similar to the C1 class of the DoD *Trusted Computer System Evaluation Criteria* (TCSEC, or "The Orange Book") [11].

2.2 CAPP at EAL3+ and EAL4+

In January 2004, SLES 8 SP 3 received an EAL3+ certification against the *Controlled Access Protection Profile* (CAPP) [19]. This certification was significant because

it marked the first time an Open Source operating system had been certified against a protection profile. Red Hat Enterprise Linux (RHEL) 4 soon followed. Both SLES 9 and RHEL 4 subsequently achieved compliance with CAPP at EAL4+.

The security capabilities evaluated for the EAL2+ certification provided an essential basis for achieving conformance on top of CAPP. The EAL2+ evaluation was essentially "CAPP Lite." Additional development had to take place in order to meet CAPP functional requirements. CAPP, which is based on the TCSEC C2 class, states that security relevant events must be audited. The lack of an audit system in Linux was the substantial impediment to meeting CAPP. In addition, CAPP requires EAL3 or higher [6].

By now, there have been several iterations of the audit system in Linux. The SLES 9 and RHEL 4 CAPP/EAL4 certification efforts implemented different audit systems. SUSE developed the Linux Audit Subsystem (LAuS) [24] for the SLES 9 certification. Red Hat developed the Lightweight Audit Framework (LAF) for the RHEL 4 certification [7]. While the two auditing systems share some common features, they have different record formats, instrumentation methodologies, and tools, and they are not compatible.

LAF is the antecedent to modern LSPP audit. LAF was submitted to the Linux Kernel Mailing List (LKML) in spring of 2004 [12] and accepted into the kernel. It was enhanced over time to meet all of the CAPP requirements, and it was finally shown to do so in the RHEL 4 certification. The auditing system used for the LSPP certification is based upon the upstream CAPP-compliant LAF implementation with numerous enhancements to meet the LSPP/RBACPP auditing requirements.

2.3 A Protection Profile Beyond CAPP

A community of Open Source security developers interested in making Linux compliant with a protection profile more stringent than CAPP formed in late 2003. There were two fairly obvious increments from which to choose: *Labeled Security Protection Profile* (LSPP) and *Protection Profile for Multilevel Operating Systems in Environments Requiring Medium Robustness* (MLOSPP) [18]. LSPP is a superset of CAPP and is based on the TCSEC B1 class. Beyond the CAPP requirements, it mainly mandates that the operating system implement Multi-Level Security (MLS)—enforcement of Bell-LaPadula rules. MLOSPP is mostly a superset of LSPP and is based on the TCSEC B2 class. In addition to the LSPP requirements, it mandates integrity labels, a cryptographic module, and a trusted path, among other features.

LSPP is supposed to be sunset 18 months after the next draft of MLOSPP is validated. Given what appeared to be the near term validation of a newer MLOSPP draft and given the sunsetting of LSPP, the community of interest initially favored meeting MLOSPP versus meeting LSPP. But it eventually became apparent that validation of the new MLOSPP draft was moving ever farther into the future. Moreover, some MLOSPP requirements, like trusted path, are universally appealing, while others are perhaps less so, such as integrity enforcement. The case for meeting MLOSPP remains unclear at present.

In addition to meeting LSPP, the community decided that meeting RBACPP would be useful. Other operating systems have set a precedent of achieving RBACPP compliance alongside LSPP [10], even though RBACPP is incomplete, somewhat vague, and unvalidated. Although the overall security goals of RBACPP are obscure, it does contain RBAC and role manipulation auditing requirements that complement LSPP fairly well. Moreover, developing features for RBACPP compliance has meant relatively little effort beyond strictly LSPP development given the built-in SELinux RBAC capabilities.

3 Meeting LSPP and RBACPP Requirements

A concerted effort among interested parties to develop an LSPP-compliant Linux began towards the latter part of 2004. Linux,as it existed then, had a number of security features, including SELinux, that met many of the LSPP and RBACPP functional requirements. However, significant gaps remained.

LSPP and RBACPP specify functional requirements derived from Part 2 of the Common Criteria [9] to address specific security areas. Broadly, LSPP contains DAC, BLP, and audit requirements, while RBACPP levies role-based access control requirements. To elucidate the features being developed by the LSPP development community, it is helpful to examine the functional requirements for each of the protection profiles and how to meet them.

3.1 Mapping Requirements to Existing Features

3.1.1 LSPP

LSPP adds a number of additional requirements to CAPP:

- Security Audit: New audit points, subject and object labels in records, the ability to search and sort based on labels.

- User Data Protection: Subject and object labels, MLS policy enforcement, new controls on the import and export of labeled and unlabeled user data, with explicit requirements for labeled print marking.
- Identification and Authentication: Addition of clearances to user attribute definition, subject sensitivity label, and enforcement of MLS policy on subject labels.
- Security Management: MLS restrictions on object label changes, static MAC attribute initialization, MLS enforcement during object revocation, and the MAC roles object modifier role.

3.1.2 RBACPP

Using LSPP as a base, RBACPP adds some additional requirements pertaining to roles, enforcement based on role membership, and management of role data:

- Security Audit: Audit of user, role and privilege changes, of roles that make actions possible, and of user session id or terminal type for all records; and the ability to search and sort and audit data based on roles enabling access.
- User Data Protection: Enforcement of the RBAC policy on subjects and objects, with permissions governed by role membership.
- Identification and Authentication: Addition of role data to user attribute definition.
- Security Management: RBAC restrictions on role management, mandate that only secure values be used, RBAC control of data, application of RBAC to revocation, admin and user security roles, and role hierarchies.
- Protection of Target of Evaluation (TOE) Functions: Preservation of the RBAC database across failures, manual recovery with maintenance mode after service interruptions, indication of recovery success or failure, and self tests that run periodically or on demand.
- TOE Access: Restriction of users to authorized roles, denial of login for users with empty role sets.

3.2 Development Roadmap

Using CAPP functionality as a base, and SELinux for labels, MLS enforcement, and MLS audit, the LSPP development community set about attempting to assess the gaps between existing features and those required to meet LSPP and RBACPP.

A specialized MLS LSM would have perhaps been the easiest way to meet the LSPP MLS requirements. However, SELinux contained many of the features required to meet LSPP:

- It provided a means of associating labels with subjects and objects on a system.
- It also provided a concise expression of MLS rules in addition to its better known Type Enforcement (TE) model.
- It assigned clearances to subjects upon their login.
- It had a built-in Role Based Access Control (RBAC) capability.
- It provided enhanced logging for security relevant events on the system.
- It had a good level of userspace integration and provided enabling tools.

Though SELinux partially met the MAC requirements, it needed a great deal of work to make MLS usable. The policy, kernel, and userspace tools had to be rebuilt to use MLS mode. The policy was monolithic, meaning that simply adding a user mapping required the administrator to edit, compile, and reload the policy. The MLS policy was virtually nonexistent and had to be completed and corrected. It was not possible to audit changes to the policy to a sufficient level of granularity.

It was, however, possible to gain good MAC audit data from SELinux Access Vector Cache (AVC) messages alone. But, because Linux Security Module (LSM) hooks are located after DAC checks, DAC denials are not auditable with LSM hooks alone. So, it was natural to continue to extend the existing LAF ptrace syscall auditing framework along with AVC messages. This also kept Linux audit orthogonal to SELinux, allowing other LSMs to use the audit system.

The audit system required a substantial overhaul to meet certification requirements. A large number of new hooks had to be placed to collect the necessary audit information. It also had to collect subject and object labels, subject role, and terminal ID for DAC and MAC permission checks. Tools had to be modified to make use of the new fields. Changes to user and role data needed to be audited. Trusted programs had to be instrumented. SELinux AVC messages had to be formatted into the standard audit record format to be easily searchable and sortable.

Devices used for import and export of labeled user data had to be controlled by the policy and had to

maintain the association between data and security labels. There was already an effort underway to add labeled IPsec to the kernel to support SELinux TE labels, which would allow inclusion of networking in the TOE. Archivers such as *star* could already preserve the SELinux filesystem labels stored in extended attributes. However, labeled networking had to be able to support MLS. CUPS had to be instrumented not only to add security marking, but to sanitize input so that security marking could not be overridden by a malicious print job.

3.3 Out of Scope Features

It was useful to exclude certain Linux features from the evaluation that would have unreasonably expanded its scope. These features have difficult problems which must be solved before they can be included in a certification. Independent software vendors have an opportunity to fill some of these requirements.

3.3.1 MLS-aware System Components

The SELinux development community put some effort into adding label support to the X Window System using the XACE security hook extension. The SELinux XACE extension adds an object manager to the X server. However, both the hooking and policy work were incomplete. Also, no Open Source window manager existed that could handle sensitivity labels.

It was a relatively easy decision to exclude NFS. NFSv4 has support for extended attributes, which NFSv3 lacks. When the LSPP development community commenced work in earnest, NFSv4 was not yet implemented. Harmonizing physical filesystem, network, and NFS labels would have been an onerous task with NFSv3. Moreover, it seemed pointless given that NFSv4 would eventually provide an elegant end-to-end labeling capability.

Other smaller system components must also be made MLS-aware to be usable in an MLS environment. For example, slocate must be modified to enforce MLS rules as well as DAC and TE, or filename downgrades will occur. Another prime example is multilevel mail. The mail server must be MLS-aware to enforce BLP restrictions based on labels.

3.3.2 Covert Channels

A covert channel is a mechanism that can be used to bypass MLS information flow restrictions [13]. Covert channels are divided into timing channels and storage channels. Timing channels require a clock or other timekeeping mechanism (such as keystroke timings or disk access latency) to exploit; storage channels do not.

LSPP does not require any analysis of covert channels, and such concerns were beyond the scope of LSPP/RBAC evaluation.

For example, here are some candidates for covert channels that are likely to be available in current systems:

- file and directory access timestamps. (Using the "noatime" mount option can mitigate this.)
- binding TCP ports fails if any process has bound the port already, independent of the MLS levels involved. (Implementing polyinstantiated network ports would help.)
- timing channels, such as scheduler latency, or disk access latency based on cache hits and misses. (A typical approach to avoid this would be assigning fixed time slices to processes instead of dynamic scheduling.)

4 Development and Lessons Learned

This section summarizes the experiences during the ongoing development of the LSPP and RBACPP extensions to the Linux kernel and userspace in general, focusing on RHEL5. The plans and roadmaps discussed in the previous section were helpful to guide development, but in several cases the end result was significantly different from the original design.

First, we briefly look at major new developments for functionality that needed to be added to meet requirements. Then we examine areas where existing tools needed to be enhanced or modified to support the new policies. Finally we give examples of areas where the current implementation encountered limitations or exposed other concerns.

4.1 New Components

4.1.1 Labeled Networking

In an MLS environment, it is essential that network data transfers are properly protected by the MLS policy. This requires a mechanism to associate the network data with MLS labels, and security checks to ensure that processes can only communicate over the network at appropriate levels. The current implementation uses *xinetd* as a MLS-aware network daemon that launches server programs (such as *sshd*) at a single MLS level that matches the incoming connection. This ensures that login sessions automatically run at the correct level in the MLS networked environment, and that *ssh* cannot be used to transfer data across levels.

Current Linux kernels provide support for two independent MLS-enhanced networking implementations:

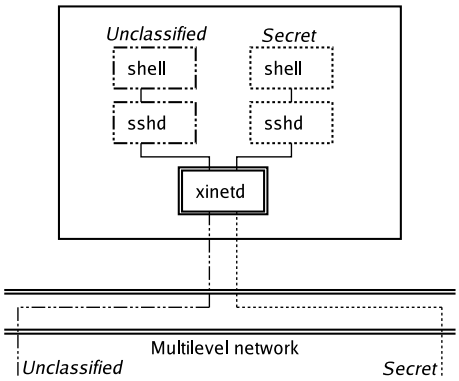

Figure 1: *ssh* login using multilevel network

NetLabel which implements the Commercial IP Security Option (CIPSO) and mlsxfrm which is based on IPSec.

In 2001, James Morris developed the SELOPT CIPSO implementation [17] which was not accepted into the Linux kernel. The NetLabel CIPSO implementation developed by HP [16] was accepted in 2006.

The main advantage of CIPSO is that it can interoperate with other MLS operating systems. On the other hand, CIPSO has some inherent limitations. It relies on a trustworthy network due to its lack of encryption and authentication, it is currently limited to IPv4, and it imposes restrictions on the usable number of categories due to packet size limits. NetLabel does not currently support TE, but this could potentially be added, for example by using FIPS-188 "Free Form" tags as in the SELOPT implementation.

The mlsxfrm implementation extends the IPSec Security Association (SA) with additional SELinux context information, and it adds extensions to key management to propagate these contexts between systems ([27], [15]). The initial implementation supported TE only. TCS added support for MLS and automatic SA negotiation using *racoon*; manual key management would not suffice due to the potentially huge number of SAs needed.

The mlsxfrm implementation is currently Linux-specific and cannot interoperate with existing MLS operating systems, but it supports the IPSec encryption and authentication features that makes it potentially suitable for use over untrusted networks. It supports both TE and MLS, and it can efficiently and arbitrarily handle many categories because the contexts only need to be transferred once when establishing the SA instead of being included in each packet.

4.1.2 VFS Polyinstantiation

Polyinstantiation is a mechanism that permits multiple versions of an object to exist in the same location in a namespace, where the specific object being accessed depends on the subject's security context. LSPP and RBACPP do not require polyinstantiation functionality, but it is very helpful for supporting applications that were not developed to work in an MLS environment.

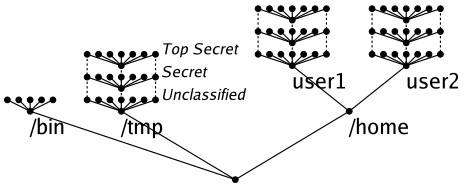

Figure 2: Polyinstantiated Directories

Legacy applications expect to be able to read and write the */tmp*, */var/tmp/*, and *$HOME* directories (e.g. for run control files), but it is impossible to permit this access for all subject levels securely when enforcing MLS rules. The polyinstantiation mechanism uses independent directories for different security levels, for example an "unclassified /tmp" and a "secret /tmp", and processes that use the path */tmp* are transparently redirected to the appropriate instance directory.

The polyinstantiation implementation based on the *unshare(2)* system call and the *pam_namespace.so* module supports flexible and locally configurable polyinstantiation, and the instance directories are dynamically created as required.

In practice, this style of polyinstantiation is unobtrusive for the temporary file directories, but the polyinstantiated home directory is very unpleasant to use. People do not expect their files to disappear depending on what settings they use to log in. In hindsight, it would have been nice if run control files used a separate directory such as *$HOME/etc/* to keep them separate from user created files, but it is too late to change that. An alternative might be to redirect access to all files matching the pattern `$HOME/.*` to a polyinstantiated subdirectory.

By default, the permissions of the polyinstantiated directories are very restrictive, and a "secret" user is not permitted to access the "unclassified" version of the home directory at all. This is a more strict than required by the MLS policy, and the *ignore_instance_parent_mode* flag can be used to permit such access.

4.2 Modifications to Existing Tools

4.2.1 Multilevel Cron

Job schedulers such as *cron*, *anacron*, and *at* run processes on a user's behalf, and must be aware of the special properties of an MLS system to work securely (or at all).

Enhancements to the cron system introduced the new `SELINUX_ROLE_TYPE` variable, which allows for the specification of multiple individual SELinux security contexts for jobs in a single crontab file. The cron daemon was also enhanced to support polyinstantiation, which was necessary to ensure that the filesystem namespace visible to the cron jobs matches those of interactive user sessions. Mail delivery was disabled for the MLS environment due to lack of a suitable multilevel mail delivery mechanism. Adding mail notification, at least for administrators, would be a useful future enhancement.

4.2.2 Audit Enhancements

The Lightweight Auditing Framework (LAF) was extended to meet the additional LSPP and RBACPP requirements, for example by adding subject and object labels, and by defining additional events such as data import/export. In general, the combination works well, but there is some mismatch between the very different format of SELinux "avc" audit records versus those generated by the syscall audit mechanism that makes it more difficult to interpret the audit trails consistently.

Future potential for audit enhancements would include a more efficient and structured binary audit record format. The flexible generator/consumer architecture of the audit daemon could be used to provide distributed audit trails.

4.2.3 Labeled Print

Labeled print was another component added to SELinux. This component is responsible for the addition of MLS labels to documents' banner pages, headers and footers. The SELinux policy enforces the required MLS restrictions for accessing printer devices.

4.3 Concerns and Open Issues

4.3.1 Critical programs not MLS aware

The SELinux philosophy appears to be largely based on the principle of keeping most applications unmodified and unaware of mandatory security features, and using the SELinux policy to add additional restrictions on top of the standard Linux DAC/capabilities policy to ensure that the applications are well behaved. In this model, adding `allow` rules or adding type attributes that grant privileges is generally not security critical, because even misconfigured rules will not make the total system security worse than if SELinux were switched off.

On an MLS system, there is an important difference: MLS information flow rules are now a fundamental part of the security policy model, and SELinux is not only adding additional restrictions, it is now the core enforcement mechanism for the MLS aspects of the security policy.

Type attributes that provide override capabilities add additional privileges to applications and allow them to break the MLS rules. This can potentially be exploited by a malicious user to circumvent the MLS restrictions. This is the "confused deputy" problem and is comparable in impact to adding "SUID root" privileges to a program that was not designed to execute with elevated privileges.

For example, the Secure Shell daemon *sshd* is a complex application that can access filesystem and network resources on a user's behalf, for example, by offering TCP connection forwarding. The implementation is unaware of the need to enforce MLS restrictions, and it depends on SELinux enforcing the information flow constraints.

However, *sshd* gains additional privileges from SELinux type attributes, including the right to completely ignore MLS constraints for filesystem access. Here is the result of an `apol` query in the reference policy [25] as included in RHEL5 beta2: [3]:

```
sshd_t (14 attributes)
    can_change_object_identity
    can_change_process_identity
    can_change_process_role
    daemon
    domain
    keyring_type
    mlsfdshare
    mlsfiledowngrade
    mlsfileread
    mlsfileupgrade
    mlsfilewrite
    mlsprocsetsl
    privfd
    ssh_server
```

The MLS read/write override privileges are separate for network and file access. Because *sshd* does not have special networking privileges, the policy properly enforces the MLS information flow restrictions for the TCP connection forwarding mechanism. The *sshd* daemon does have the right to ignore other MLS restrictions, for example, for file read and write operations, even though some of these privileges are only needed for very specific purposes. The policy and the program code are developed separately and must work together closely to ensure a secure system.

It would be helpful if privileges could be restricted to those code paths that actually need them, for example, many of *sshd*'s privileges are only needed within the PAM library, for example, for setting up polyinstantiation.

The *sshd* example was intended to illustrate the potential danger of adding privileges to applications that are not aware of the need to enforce policy themselves because they cannot depend on the OS to do it for them. Implementing that enforcement in the application results in code duplication that is potentially error prone and undesirable. The principle of least privilege suggests minimizing the additional privileges given to applications, but this is not always easy due to the need to keep legacy applications working.

As a related issue, while the *apol* tool can be used to see the additional privileges, the refpolicy abstraction layers make it unobvious where the override privileges are coming from, and they are not easily visible when examining the *ssh.** policy source files.

For example, the following types in the reference policy have the *mlsfileread* override capability:

> *NetworkManager_t auditctl_t auditd_t bootloader_t consoletype_t crond_t cupsd_t dmidecode_t fsadm_t fsdaemon_t getty_t hald_t init_t initrc_t iptables_t klogd_t kudzu_t load_policy_t local_login_t logrotate_t mount_t newrole_t pam_console_t quota_t readahead_t remote_login_t restorecon_t rpm_script_t rpm_t semanage_gui_t semanage_t setfiles_t setrans_t sshd_extern_t sshd_t tmpreaper_t udev_t*

The programs running in those domains cannot rely on the OS to prevent MLS information leaks due to the override. In recent policies, the restrictions on permitted transitions ensure that almost all of these domains are inaccessible to unprivileged user domains such as *user_t*, leaving only the programs *chage, passwd, newrole,* and *su* accessible with MLS override privileges. In previous versions of the policy, many more programs were accessible.

4.3.2 Newrole and PTY information flow

During development, an ugly issue was that pseudoterminals (PTYs) did not implement SELinux checks for communication between the master and slave ends of the PTY. This causes difficulties when combined with the ability of *newrole -l* to run processes at a different MLS level. It is easy to run a *newrole* session in a PTY (similar to the TCL *expect* program), with the two ends of the PTY connected to processes that are running at different MLS levels. The driver program can then capture output data that a high sensitivity process sends to the PTY slave end, and write it to disk at its low sensitivity level.

Enforcing the data flow restrictions in the PTY driver would be difficult to implement without breaking existing applications.

The workaround for this issue was to prevent normal users from using the privileged *newrole -l* program to change MLS levels while communicating on a PTY device. The *newrole* program is one of the few trusted programs that has the right to change the active MLS level (corresponding to the `mlsprocsetsl` attribute), and adding a restriction to *newrole* closes the information leak.

4.3.3 Executable Type Transitions

SELinux supports automatic domain transitions by assigning **_exec_t* object types to executable files for which the policy defines transition rules to specific target domains.

The implementation of this mechanism exposes potential security flaws, environment contamination, and a race condition for script files.

Environment contamination occurs when an unprivileged user executes a program running with higher privileges, and user modifications to the execution environment change the behavior of the privileged program. This is the well-known issue faced by SUID programs [26], and the same protection mechanisms apply. The kernel activates "atsecure" mode when executing SUID programs or type-transitioning executables. This mode enables security mechanisms in GLIBC, for example, ignoring the LD_PRELOAD environment variable.

The "atsecure" mechanism is not sufficient to protect script files because script interpreters may be influenced by other environment variables such as PYTHONPATH or PERLLIB. To protect against this, scripts must use interpreter-specific protection features such as Perl's "taint" mode (`-T` flag) or Python's `-E` flag, and these arguments must be provided on the `#!` line so that they are processed by the kernel. The usual Python idiom of using `#!/usr/bin/env python` is unsafe because the *env* program uses the PATH environment variable to locate the interpreter binary.

The other issue is a race condition between time of check and time of use for executable script files. The kernel assigns the new domain after reading the file label, and then it executes the interpreter. The interpreter separately opens the script file, which may now be a different object.

Both of the security flaws are only relevant if the new domain has higher privileges than the source domain. Most of the transitions in the SELinux policy are designed to remove privileges, but some deserve further analysis, such as *rpm_t*, which has more privilege attributes than *sysadm_t*.

4.3.4 User and Role Management

RBACPP requires a "hierarchical roles" feature that allows administrators to define roles in terms of other roles. SELinux supports a "dominates" operator [23] that provides part of this functionality.

The "dominates" relationship implies that the dominating role is permitted to associate with the union of types associated with the dominated roles and permits changing roles to the dominated roles, but it does not grant any privileges to the default type associated to the role.

As an example, define a new role *root_r* with default type *root_t*, which dominates the *sysadm_r*, *secadm_r*, and *auditadm_r* roles. A *root_r* user may freely change to any of the types permitted for the other roles, but by default does not have permission to perform any actions directly that require the privileges of the other roles.

In practice, when the administrator wants to perform a sysadm action, it is necessary to use the command:

```
newrole -t sysadm_t
```

which is not a big improvement over the situation without hierarchical roles:

```
newrole -r sysadm_r
```

Of course, it is possible to define TE rules to give rights to the *root_t* type, but this would require duplication of much code from the appropriate refpolicy modules. A more powerful type inheritance mechanism would be helpful, and this could be implemented within the policy tools and would not require any changes to the core mechanisms.

The RBACPP also requires a mechanism that allows administrators to define new roles and to assign privileges to roles. The modular SELinux policy is an obvious candidate for meeting this requirement, but it is important to ensure that locally added roles do not undermine the general security policy.

The MLS policy is implemented with constraint rules combined with well-defined override capabilities, and these constraints cannot be weakened or modified by policy modules. This provides good assurance that the MLS restrictions will remain intact when defining or modifying roles.

Administrators can define new administrative roles that use the override mechanisms to have additional access rights including ignoring the MLS restrictions. For example, a "backup administrator" role could provide read-only access to any file on the system, but would not provide the right to write or modify any files beyond those to which the user normally has access.

RBACPP also requires that administrators can assign and revoke object access rights to roles. In the SELinux implementation, this is possible indirectly. It requires assigning appropriate object types to the objects, defining TE access rules for domain types, and associating the domain types with roles. This meets the baseline requirement, but complex RBAC policies would benefit from a more direct mechanism. On the other hand, it is unclear if assigning direct role-based object access rights is useful in practice, and the more powerful TE rules are likely to be the mechanism of choice for defining access rules.

5 Conclusions

The MLS and TE capabilities of SELinux provided much of the basic functionality needed to meet LSPP and RBACPP requirements. Using SELinux as a starting point, various features and modifications needed to be developed to allow legacy applications to be used in a restrictive MLS environment. The task proved to be more complex than initially thought, and despite the early plan and design efforts, the final outcome did not always match the original expectations. Currently, development is continuing and the parties involved managed to find common grounds to make this certification possible. In some cases that required the exclusion of certain features that were considered problematic while in others, it meant imposing additional restrictions on certain applications to remediate potential security problems. Now that certification efforts are nearing completion, it is apparent that many enhancements could have been made if time and resources permitted. However, as with previous certifications, the lessons learned from these efforts will be taken into consideration and will be useful in future work.

The Common Criteria certification efforts equipped Linux with additional security features that allow its use in a range of environments for which it had previously not been suitable. MLS is primarily relevant for military and government installations, and RBAC can help medical and financial institutions meet strict data processing requirements. Moreover, having these features available in a general purpose operating system has the potential of improving security for other uses as well.

6 Acknowledgments

The work described in this paper was a community development effort by Red Hat, IBM, HP, Tresys, the NSA, and the many participants from the redhat-lspp [2], linux-audit [1], and selinux [4] mailing lists who contributed code, documentation, helpful advice, and constructive criticism.

This work represents the view of the author and does not necessarily represent the view of IBM and/or atsec.

IBM, IBM (logo), e-business (logo), pSeries, e (logo) server, and xSeries are trademarks or registered trade-

marks of International Business Machines Corporation in the United States and/or other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.

7 Availability

Red Hat Enterprise Linux 5 is available from the Red Hat Web site: `http://www.redhat.com`

Preliminary versions of the configuration script used to set up the evaluated configuration are temporarily available from: `http://klaus.vh.swiftco.net/lspp/`

Final versions of the script will be released by Red Hat after CC evaluation is complete.

References

[1] Linux Audit Mailing List. `http://www.redhat.com/mailman/listinfo/linux-audit`.

[2] Redhat LSPP Mailing List. `http://www.redhat.com/mailman/listinfo/redhat-lspp`.

[3] RHEL5 Beta2 SELinux Reference Policy. `selinux-policy-2.4.6-15.el5`.

[4] SELinux Mailing List. `http://www.nsa.gov/selinux/info/list.cfm`.

[5] ATSEC GMBH AND IBM CORPORATION. *SuSE Linux Enterprise Server V 8 Security Target, v1.6*. 2003. `http://www.commoncriteriaportal.org/public/files/epfiles/0216b.pdf`.

[6] ATSEC GMBH AND IBM CORPORATION. *SuSE Linux Enterprise Server V 8 with Service Pack 3 Security Target for CAPP Compliance, v2.7*. 2003. `http://www.commoncriteriaportal.org/public/files/epfiles/0234b.pdf`.

[7] ATSEC GMBH AND IBM CORPORATION. *Red Hat Enterprise Linux Version 4 Update 1 Security Target for CAPP Compliance, v2.6*. 2005. `http://www.commoncriteriaportal.org/public/files/epfiles/ST_VID10072-ST.pdf`.

[8] BELL, D. E., AND LAPADULA, L. J. Secure Computer Systems: Unified Exposition and MULTICS Interpretation, 1976. `http://csrc.nist.gov/publications/history/bell76.pdf`.

[9] COMMON CRITERIA COPYRIGHT HOLDERS. *Common Criteria Part 2: Security functional requirements, Version 2.1*. 1999. `http://www.commoncriteriaportal.org/public/files/ccpart2v21.pdf`.

[10] COMMONCRITERIAPORTAL.ORG. List of Evaluated Products. `http://www.commoncriteriaportal.org/public/consumer/index.php?menu=4`.

[11] DEPARTMENT OF DEFENSE. *Trusted Computer System Evaluation Criteria, DoD Standard 5200.28-STD*. 1985. `http://csrc.ncsl.nist.gov/publications/secpubs/rainbow/std001.txt`.

[12] FAITH, R. [PATCH][RFC] Light-weight Auditing Framework, 2004. `http://lkml.org/lkml/2004/3/1/125`.

[13] GASSER, M. *Building a Secure Computer System*. Macmillan of Canada, 1988. `http://nucia.ist.unomaha.edu/library/gasserbook.pdf`.

[14] IBM CORPORATION AND SUSE. IBM and SuSE Linux Earn First Security Certification of Linux, 2003. `http://www-03.ibm.com/press/us/en/pressrelease/5662.wss`.

[15] JAEGER, T., KING, D., ET AL. Leveraging IPsec for Distributed Authorization, 2006. `http://nsrc.cse.psu.edu/tech_report/NAS-TR-0037-2006.pdf`.

[16] MOORE, P. NetLabel - Explicit labeled networking for Linux. `http://netlabel.sourceforge.net`.

[17] MORRIS, J. Labeled IPv4 Networking for SELinux (selopt), 2001. `http://lwn.net/2001/1213/a/selopt.php3`.

[18] NATIONAL SECURITY AGENCY, INFORMATION ASSURANCE DIRECTORATE. *Protection Profile for Multilevel Operating Systems in Environments Requiring Medium Robustness, Version 1.22*. 2001. `http://www.commoncriteriaportal.org/public/files/ppfiles/PP_MLOSPP-MR_V1.22.pdf`.

[19] NATIONAL SECURITY AGENCY, INFORMATION SYSTEMS SECURITY ORGANIZATION. *Controlled Accesss Protection Profile, Version 1.d*. 1999. `http://www.commoncriteriaportal.org/public/files/ppfiles/capp.pdf`.

[20] NATIONAL SECURITY AGENCY, INFORMATION SYSTEMS SECURITY ORGANIZATION. *Labeled Security Protection Profile, Version 1.b*. 1999. `http://www.commoncriteriaportal.org/public/files/ppfiles/PP_LSPP_V1.b.pdf`.

[21] REYNOLDS, J., AND CHANDRAMOULI, R. *Role-Based Access Control Protection Profile, Version 1.0*. 1998. `http://www.commoncriteriaportal.org/public/files/ppfiles/RBAC_987.pdf`.

[22] SHANKAR, K. S., AND KURTH, H. Certifying Open Source–The Linux Experience. *IEEE Security & Privacy 2*, 6 (2004), 28–33. `ftp://www6.software.ibm.com/software/developer/library/os-ltc-security/IEEE%20Article.pdf`.

[23] SMALLEY, S. Re: hierarchical roles. `http://marc.theaimsgroup.com/?l=selinux&m=115497265200266&w=2`.

[24] SUSE LINUX AG. Linux Audit-Subsystem Design Documentation for Linux Kernel 2.6, v0.1, 2004. `http://www.uniforum.chi.il.us/slides/HardeningLinux/LAuS-Design.pdf`.

[25] TRESYS TECHNOLOGY. SELinux Reference Policy. `http://oss.tresys.com/projects/refpolicy`.

[26] WHEELER, D. A. Secure Programming for Linux and Unix HOWTO. `http://www.dwheeler.com/secure-programs/`.

[27] YEKKIRALA, V. Granular IPSec associations for use in MLS environments. `http://www.redhat.com/archives/redhat-lspp/2006-June/msg00067.html`.