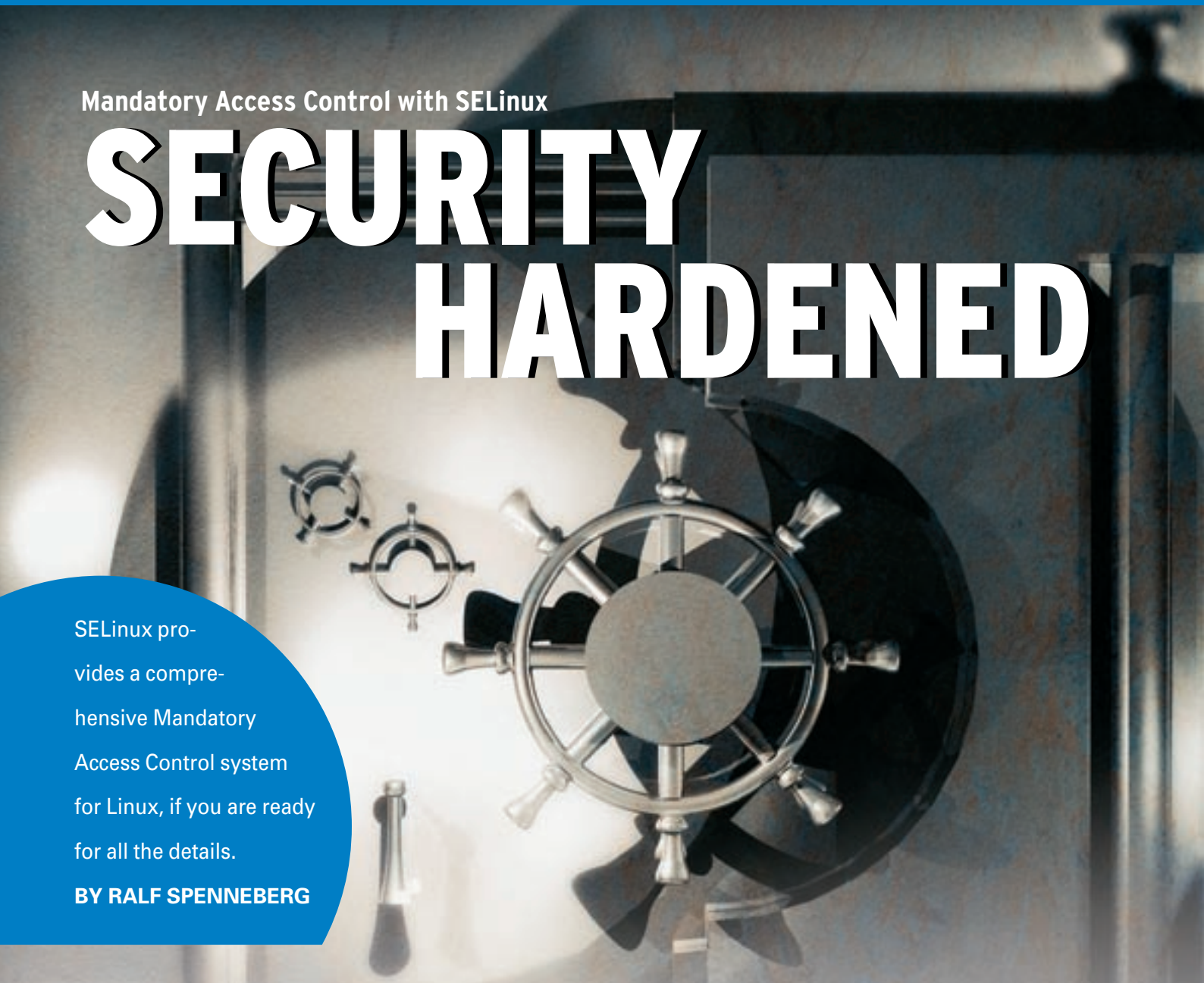

Mandatory Access Control with SELinux

# SECURITY HARDENED

SELinux provides a comprehensive Mandatory Access Control system for Linux, if you are ready for all the details.

**BY RALF SPENNEBERG**

SELinux is a security-enhanced adaptation of the Linux kernel developed under the auspices of the US National Security Agency (NSA). According to the NSA, SELinux works by enforcing "access control policies that confine user programs and system servers to the minimum amount of privilege they need to do their job."

The security of an ordinary Linux system is based on a concept known as Discretionary Access Control (DAC). In a DAC system, a user is granted access to a resource (such as a file or directory) based on the user's credentials, and users have the discretion to modify permissions for any resources they happen to control. This design gives attackers a means for gaining entry to a system. If root launches the Adobe Reader to access a PDF from an untrusted source, an attacker could exploit a vulnerability to start a root shell, even though root shells have nothing to do with what Adobe Reader is supposed to be doing.

SELinux and other similar security systems replace DAC security with a much safer alternative known as Mandatory Access Control (MAC). A MAC system lets the system administrator define policies that provide granular control over a user's ability to access resources and modify permissions. SELinux also maintains control over the range of privileges assigned to a process. A process is not allowed to possess privileges it does not need.

If SELinux is implemented properly, an intruder who gains access to a Linux system will be severely limited in what they can accomplish once they get inside. As you'll learn in this article, SELinux provides a very powerful and verstatile security framework – if you're willing to contend with the complexity of getting your system properly configured.

## Security Context

SELinux uses the security *context* as the central access criterion. The context consists of the SELinux user account (not to be confused with the Unix user), along with the user's role and type.

SELinux stores the security context for files directly on the filesystem, although this approach only works with Ext 3 and XFS as of this writing. Patches are available for Reiser FS, but JFS is not supported yet.

Calling *ls -Z* gives you a file's security context: Figure 1 shows *user_u:object_r:paper_t*. The Unix user and group are *spenneb*, whereas SELinux identifies the user as *user_u*; the user's role is *object_r*, and the type is *paper_t*. The suffixes for these names indicate the context component. Processes are also assigned a secu-


**Figure 1: SELinux assigns all files and processes to a security context. The context and the policy decide what kind of access is allowed.**

rity context, which you can output by running *ps -Z* (see Figure 1).

## Deny All

One of the principles of SELinux is that each process is associated with a *domain* and each object is associated with a *type*. SELinux starts off by denying any kind of access flat out. Even access by the *example_t* domain to the *example_t* type is prohibited. Every action has to be explicitly allowed. The policy might look like this for the *emacs_t* domain:

```
allow emacs_t paper_t:file
{create ioctl read getattr ➥
lock write append setattr};
```

This allows processes in the *emacs_t* domain to perform the operations in the curly braces on *paper_t* type objects. The *:file* object class specifies that these objects must be regular files.

## Changing Domains

If a normal user launches the Emacs editor, the process does not automatically use the *emacs_t* domain. In fact, permissions are required to change domains, and the process is automated. To allow this to happen, a rule first defines an entry point to the new domain for the Emacs editor binary. The program file needs the *emacs_exec_t* type (as a filesystem label) and SELinux needs the following rule:

```
allow emacs_t emacs_exec_t:➥
file entrypoint;
```

SELinux will now allow *emacs_exec_t* type executables to enter the *emacs_t* domain when launched. So far so good, but SELinux still has to allow the caller to perform this action. As the caller resides within another domain, a domain change occurs when the process is started. An allow rule, and a type transition rule, are needed in order for this to work:

```
allow { userdomain } ➥
emacs_t:process transition;
type_transition { userdomain }➥
 emacs_exec_t:process emacs_t;
```

The *allow* rule allows all processes belonging to a domain on the *userdomain* list to launch new processes in the *emacs_t* domain. The type transition rule that follows specifies that the called process will be assigned to the *emacs_t* domain, rather than inheriting the caller's domain, if the program file is of the *emacs_exec_t* type. And this only applies if the caller belongs to a *userdomain*.

To allow SELinux to deploy these rules on a system, all files must be assigned an appropriate security context in a process known as labeling. **.fc* or file context files specify which file receives which label:

```
/usr/bin/emacs(.*) -- ➥
system_u:object_r:➥
emacs_exec_t
```

When a user somewhere on the system creates a file that matches the pattern, SELinux assigns the specified *emacs_exec_t* type context.

## Complexity

This example clearly demonstrates the enormous granularity of SELinux, and the ensuing complexity. A part-time administrator has no chance of keeping track. An SELinux policy can easily reach a size of 6MB – rules only, and in ASCII. Developers have been trying to do something about the complexity issue for quite awhile now.

Fedora Core is one of the leading SELinux distributions. Up to version 4, Fedora had 2 policies: Strict and Targeted. Whereas the Strict policy actually implemented a MAC system for every process on the Linux system, the Targeted policy simply inspected a few services that handle potentially critical data – mainly network services. The remaining processes on Fedora Core would then run in a special *unconfined_t* domain that is no different from a system without SELinux.

Processes in the Unconfined domain are allowed more or less any kind of access, as only the system's normal DAC privileges are operational. SELinux's approach is similar to the AppArmor approach in this mode. However, SELinux on Fedora Core 4 still needs a 2.5MB ASCII policy in Targeted mode to support the required levels of granularity.

## Management

Their monolithic structure makes policies difficult to manage. Changes and additions to the policy always mean changing the source code, and then rebuilding the binary. Of course, this type of change is for system administrators only.

To make things easier on system administrators, policies can contain Boolean variables whose values can change at runtime. For example, Fedora Core 4's Targeted policy has no less than 95 variables that define whether SELinux moni-

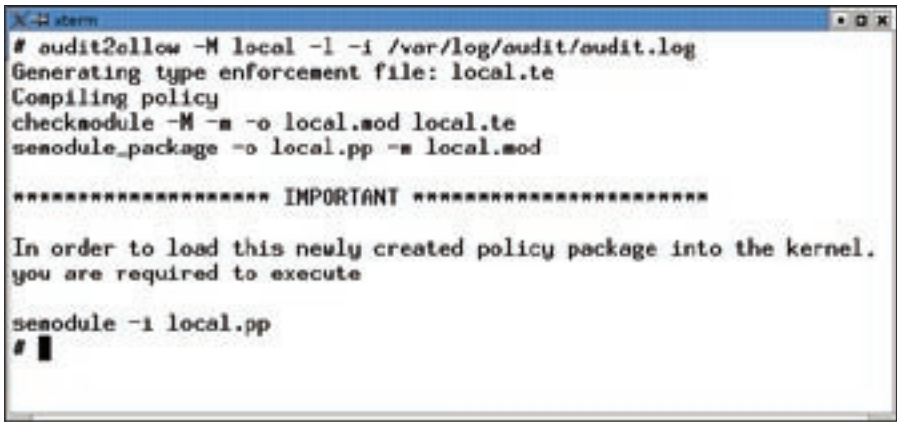


**Figure 2: audit2allow tells SELinux to analyze audit trails when events have occurred that infringe against the policy.**

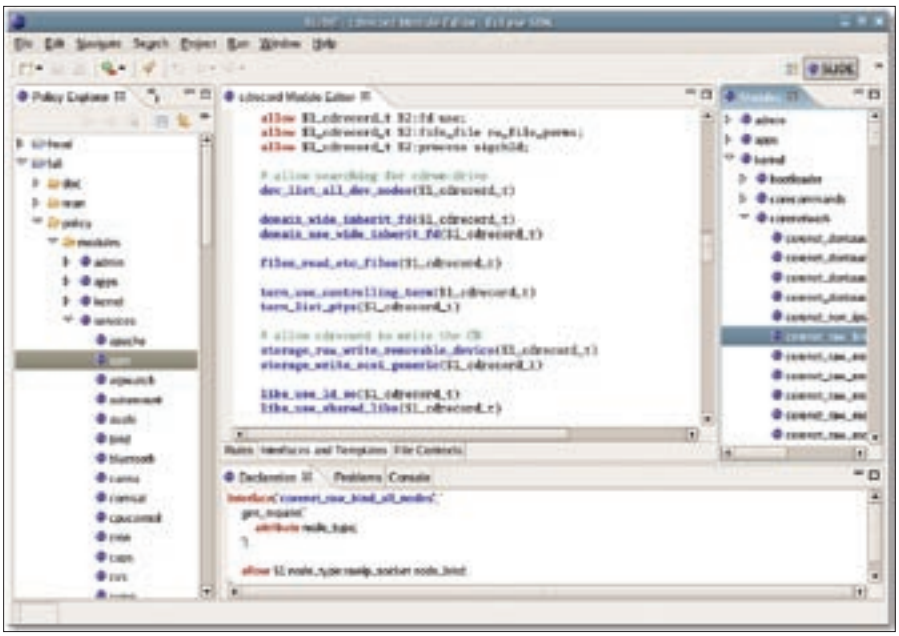

**Figure 3: The SELinux IDE, Slide, provides access to the modular SELinux Reference Policy. Slide is implemented as an Eclipse plugin.**

tors a service and how that monitoring is going to take place. For example, the *httpd_can_network_connect* variable specifies whether the web server is allowed to establish network connections of its own – to a database server, for example. Boolean variables will not let you delegate individual administrative tasks to other users.

## Reference Policy

The distinction between the Strict and Targeted policies poses a problem: it is more or less impossible to exchange rules between policies. Tresys developed the SELinux Reference Policy to address this issue. The SELinux Reference Policy has many goals, but its major concern is a high level of modularity. The idea is to give admins the ability to load, unload, and replace modules at runtime. To support communication between modules, each module defines its own interface. To improve usability and make modules easier to understand, each module comes with its own interface documentation. You can generate both a Strict policy and a Targeted policy from the reference policy. Fedora Core 5 was the first distribution to introduce this new technology.

Whereas the Targeted policy only has two modules, *base.pp* and *enableaudit.pp*, the Strict policy comprises no less than 149 binary modules. Taking dependencies into consideration, the *semodule* loads these modules.

## New Modules

A typical problem that occurs when using SELinux, is that of generating a policy for a new program. Extensions like this are now possible with binary modules, and this removes the need to pass the whole reworked ruleset to SELinux. A module comprises up to three files:

- modul.fc
- modul.te
- modul.if

The FC file contains the file contexts that define how SELinux will label individual files. The TE file contains the type enforcement rules; the IF file is new, and contains the module interface definition and accompanying documentation. Although administrators can write these files from scratch, Fedora Core 5 has a script called *policygentool* to help simplify the process.

The policy generation tool expects the system administrator to provide

## Multi-Level Security

Although Multi Level Security is one of SELinux's core design features, MLS was tagged as experimental for many months. The Reference Policy is the first MLS policy and supports the Bell-La-Padula model, which was invented in 1973 to help keep military secretes. Basically, the model assigns scope and capabilities to subjects, whereas objects have scope and rankings.

The model ensures that an object within its own scope can only be read by subjects with identical or superior capabilities. Objects can only be written if the subject has an identical or lesser ranking. When a person with a higher rank creates a file, this file can only be read by persons with identical or superior clearance to prevent sensitive data falling into the wrong hands. To implement MLS, the security context has two additional parameters: MLS Levels from *s0* through *s15* and MLS ranges from *c0* through *c255*.

### Multi-Category Security

Since almost nobody needs an MLS design apart from armed forces and secret services, the developers added the concept of Multi-Category Security (MCS) to their reference policy. MCS softens up the MLS functionality. All objects are assigned to MLS level *s0*. The system administrator can then freely select the categories of other objects. To back up the numbers with intuitive descriptions, administrators can even assign names to these categories in the *setrans.conf* file:

```
s0:c0=Confidential
s0:c1=Development
s0:c2=HumanResources
```

After you assign names to the categories, the *chcat* command understands the names as well as the categories and can assign them to files:

```
chcat -- +Confidential ➥
/tmp/file.txt
```

Whereas the file previously had a security context of *root:object_r:tmp_t:*, it now has a context of *root:object_r:tmp_t:Confidential*. The *chcat* command can also assign a users to a category. Every file the user creates from this point on will have that user's MCS category. Users without a category assignment are not allowed to access the file. Whereas this function is similar to Posix style ACLs (Access Control Lists), MCS will have more capabilities in future. For example, a printing system might issue a warning when printing a specific document, depending on the category. An email client might refuse to send documents from a specific category.

the name of the module to create, along with the path to the program the module governs. The tool then goes on to prompt the admin for a few answers before creating the configuration files and the finished module.

If you prefer the option of creating the module manually from the three files, you can run *checkmodule* to build the TE file, and then use *semodule_package* to build the *foo.pp* module file, which you can then load by running *semodule -i foo.pp*.

## Training Mode

Instead of attempting to manually compile the details of a policy, it makes sense to create rules based on audit messages logged for denied access. And this is exactly what the *audit2allow* command does; in fact it implements a kind of training mode. The results are SELinux rules that allow exactly the types of access that the tool prohibited and logged.

Sensibly, the *audit2allow* command runs in SELinux's permissive mode, in which the system simply logs policy infringements without denying them. A version of the *audit2allow* command, which has been modified for the reference policy, also supports direct module generation Figure 2 shows *audit2allow* in action.

If Auditd is not running, SELinux sends messages to */var/log/messages*. The *semodule -i local.pp* then loads and runs the new module. Manual changes to the TE file are possible, and sensible, although it is a good idea to check the results carefully.

### IDEs

A number of GUIs are available for SELinux management and administration . Slide (Figure 3) is the first GUI for the Reference Policy. Tresys Technology implemented the IDE as an Eclipse plugin that supports syntax highlighting, wizards, and auto-completion of predefined labels in the module interface. The installation requires the Eclipse SDK 3.1, the Reference Policy, and the SE Tools. The easiest way to install Slide is via an RPM or directly using Eclipse from the webpage [8].

After the install, you should have a clear-cut and tidy IDE which has a few weak points but still makes the job much easier, despite the early version number of 0.1.0.

## MAC by MAC

As of this writing, only the system administrator can change and reload SELinux policies. Whereas this was the only approach the monolithic design of the older technology supported, the new Reference Policy with its modular structure gives admins the ability to assign different access attributes to policy module files.

Thus far, a process with the ability to load a policy also had the ability to modify the whole policy. The modular structure now supports a new approach, for which Tresys Technology developed an SELinux Policy Server. Besides the targets already mentioned, the policy server looks to achieve two other goals: to support new objects managed by userspace applications, including databases and database tables. Additionally, the policy server will support an improved infrastructure for central management of policies on multiple systems.

The initial preview version already supports the idea of policies allowing specific changes to themselves. The developers introduced the new *policycon* command that applies a security context to indvidual policy modules. It is then possible to allow your own policy to change itself. Again, the *semodule* command handles this, communicating with the policy server in userspace to do so.

The programmers still advise against running a policy server in production environments, and unfortunately, things have quietened down on the development stakes.

## Development

A fairly impressive user and developer community has grown up around SELinux. The SELinux Symposium [7] takes place every two years; this year's symposium took place in February, with 29 lectures and 5 tutorials on SELinux over the course of three days. The lecture slides are available from the symposium's homepage.

Current development work is focusing on adding the IPsec frameworks to SELinux. This would allow SELinux to monitor data traffic on the network. Some of these developments have found their way into the 2.6.16 kernel [10].

Many administrators are worried about introducing vulnerabilities by using SELinux. The enormous policy, which is difficult to comprehend in its entirety, contributes to this feeling of uncertainty. However, a MAC system like SELinux is bound to enhance your system's security if you configure it properly. And remember, SELinux can't give a process privileges that it wouldn't have without a MAC. SELinux is not solely responsible for allowing or denying access; the normal Linux DAC also has to approve the access type based on legacy criteria.

We'll hope that the SELinux developers will continue working hard on usability. SELinux developers have already laid the foundations for a more usable system with more modularization and the policy server. ■

### INFO


[1] NSA website on SELinux: *http://www.nsa.gov/selinux/*

[2] SELinux for distributions: *http://selinux.sourceforge.net*

[3] SELinux Reference Policy: *http://serefpolicy.sourceforge.net*

[4] Tresys: *http://www.tresys.com*

[5] SLIDE: *http://selinux-ide.sourceforge.net*

[6] SELinux policy server: *http://sepolicy-server.sourceforge.net*

[7] SELinux Symposium: *http://www.selinux-symposium.org*

[8] SLIDE installation: *http://www.tresys.com/files/eclipse-update/*

[9] MCS introduction: *http://james-morris.livejournal.com/8228.html*

[10] SELinux for IPsec: *http://marc.theaimsgroup.com/?l=linux-netdev&m=113234097011133&w=2*



THE AUTHOR

Ralf Spenneberg is a freelance Unix/Linux trainer, consultant, and author. Ralf's business, OpenSource Training, offers training and consultancy services. Ralf has published a number of books on the topics of intrusion detection and virtual private networks.