Adding Ramparts to Your Bastille
An Introduction to SELinux Hardening

Def Con 24

Packet Capture Village

Jay Beale

Author Bio: Jay Beale

Jay Beale is the CTO and COO for the security consultancy InGuardians. He wrote Bastille Linux, the Center for Internet Security's first Unix Scoring Tool as well as its Linux Benchmark hardening guide, columns and articles for Information Security Magazine, Security Portal, and Security Focus, as well as a number of books, including those in the Jay Beale's Open Source Security Series. Jay has been invited to speak and chair conferences around the world.

SELinux Introduction

SELinux adds Mandatory Access Control (MAC) to the standard Discretionary Access Control (DAC).

Mandatory access control is defined by the system owner, preventing file/component owners from altering the access control policy.

More importantly to us, it isolates components of the system from each other.

DAC, MAC, and the AVC

Any policy decision made by the kernel first is turned over to the default Discretionary Access Control (DAC) system: Linux file permissions, ACL's, capabilities, ...

If and DAC permits the access, SELinux gets a shot.

SELinux checks its cache, the Access Vector Cache (AVC), for past answers, making the answer based on its default-deny policy if the answer isn't in the AVC.

Targeted Policy

SELinux’s most popular policy is the “targeted” policy.

It confines many of the programs started on boot, particularly those listening on the network, as well as privileged programs like Set-UID programs.

User processes generally aren't targeted by default.

This talk focuses on the targeted policy exclusively.

SELinux MAC Types

SELinux enables three major types of mandatory access control:

- Type Enforcement (TE)
- Role-based Access Control (RBAC)
- Multi-Level Security (MLS)

The targeted policy basically uses Type Enforcement exclusively.

Investigating SELinux Modes

- `getenforce` – indicates mode (permissive, enforcing)
- `setenforce <0|1>` - turn enforcing off/on until reboot
- `sestatus` – get full status information

```
# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      28
```

Persistently Setting Mode

Survive reboots via /etc/selinux/config:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of three two values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy. Only selected
#                processes are protected.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

SELinux Core Mechanism

SELinux labels processes and files with security contexts.

The policy describes how processes are allowed to interact with files, by specifying rules governing the interaction of types.

The parsing is default-deny.

This is mediated by unconfined types.

SELinux Labels

Security contexts for files come via labels:

- User
- Role
- Level
- Type

```
# ls -lZ /etc/httpd/conf
-rw-r--r--. root root
system_u:object_r:httpd_config_t:s0 httpd.conf
```

Investigating Security Context

Try out these two commands:

```
ps -eZ
id -Z
```

Examples from ps -eZ:

system_u:system_r:**avahi_t**:s0 778 ? 00:00:00 avahi-daemon
system_u:system_r:**sshd_t**:s0-s0:c0.c1023 1249 ? 00:00:00 sshd
system_u:system_r:**crond_t**:s0-s0:c0.c1023 1264 ? 00:00:01 crond
system_u:system_r:**crond_t**:s0-s0:c0.c1023 1265 ? 00:00:00 atd

SELinux Users

By default, the targeted policy doesn't confine users or roles.

```
# seinfo -u
Users: 8
   sysadm_u
   system_u
   xguest_u
   root
   guest_u
   staff_u
   user_u
   unconfined_u
```

```
# cat /etc/selinux/targeted/seusers
…
system_u:system_u:s0-s0:c0.c1023
root:unconfined_u:s0-s0:c0.c1023
__default__:unconfined_u:s0-s0:c0.c1023
```

```
# semanage login –l
Login Name          SELinux User        MLS/MCS Range        Service

__default__         unconfined_u        s0-s0:c0.c1023       *
root                unconfined_u        s0-s0:c0.c1023       *
system_u            system_u            s0-s0:c0.c1023       *
```

SELinux Roles

By default, only two roles are in play: system and unconfined.

```
ps -efZ | grep -v system_r | grep -v unconfined_r
```

If you'd like to see how to use roles to confine users, consult Red Hat's SELinux manual.

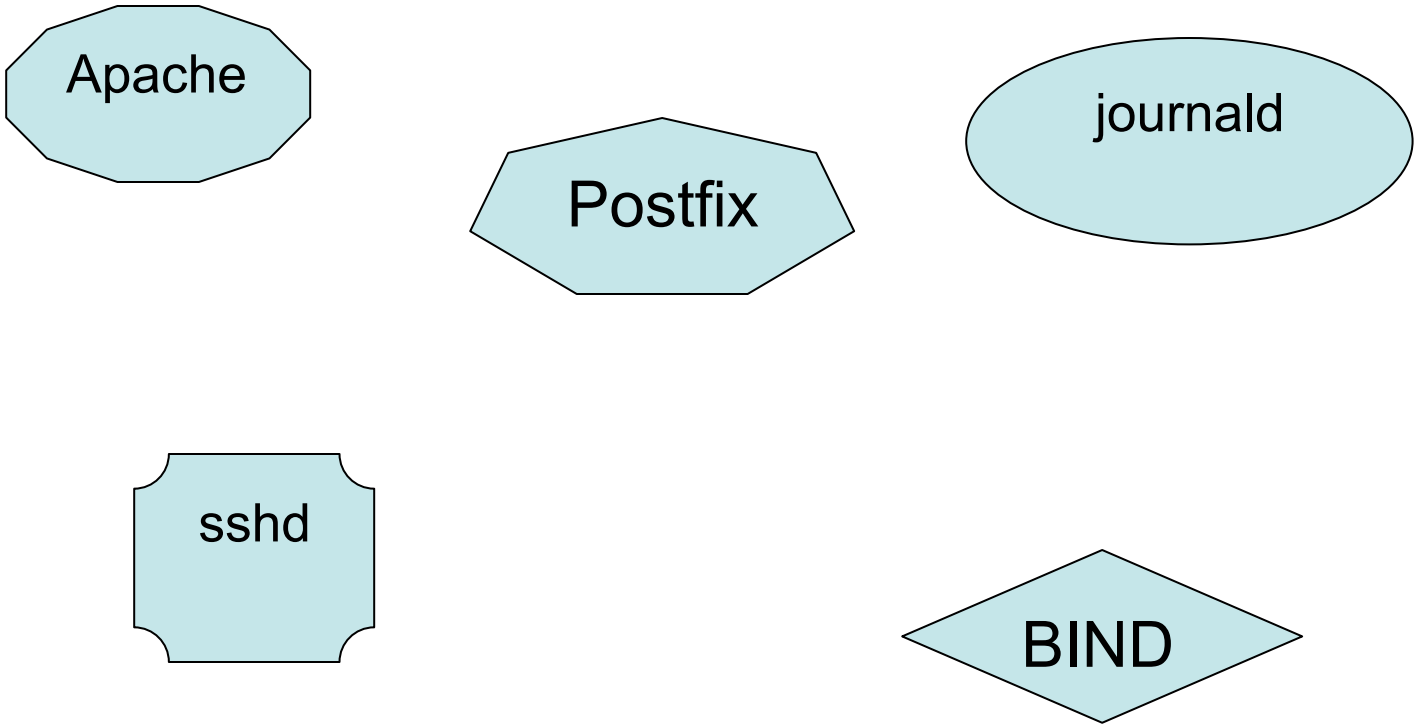
SELinux Types

The targeted policy is basically about type enforcement.

Types are also called "domains."

In particular, subjects of actions (processes, users) are said to be in Domains, while objects of actions (files, ports, sockets) are said to have Types.

SELinux Domains Diagram



SELinux User, Role, Type

Investigate the set of users with seinfo –u.

Highlight: `Users: 8`

Investigate the set of roles with seinfo –r.

Highlight: `Roles: 14`

Investigate the set of types with seinfo –t.

Highlight: `Types: 4622`

Clearly, the targeted policy puts a heavy emphasis on types.

Unconfined Types

By default, users' processes run unconfined by MAC.

```
# ps -eZ | grep unconfined
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 35675 pts/1 00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 5918 ? 00:00:00 evolution…
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 5920 ? 00:00:00 gconfd-2
…
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 5940 ? 00:00:00 seapplet
# ps -eZ | grep unconfined | wc -l
53
```

A type doesn't have to be named "unconfined" – this unconfined_t type is just an arbitrary name.

Other Unconfined Types

`unconfined_t` isn't the only unconfined domain/type.

```
# seinfo -aunconfined_domain_type -x | head -5
   unconfined_domain_type
      sosreport_t
      bootloader_t
      devicekit_power_t
      virt_qemu_ga_unconfined_t

# seinfo -aunconfined_domain_type -x | wc -l
89
```

Set-UID and Type Transitions

Just because SELinux doesn't confine users doesn't mean it doesn't constrain Set-UID programs.

Look what happens when user jay runs the passwd command:

```
$ passwd
Changing password for user jay.
Changing password for jay.
(current) UNIX password:

# ps -eZ | grep passwd
unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023 root 54891 54764  0 23:58 pts/2 00:00:00 passwd
```

There's a kind of magic here, called type transitions.

Example: Type Transitions

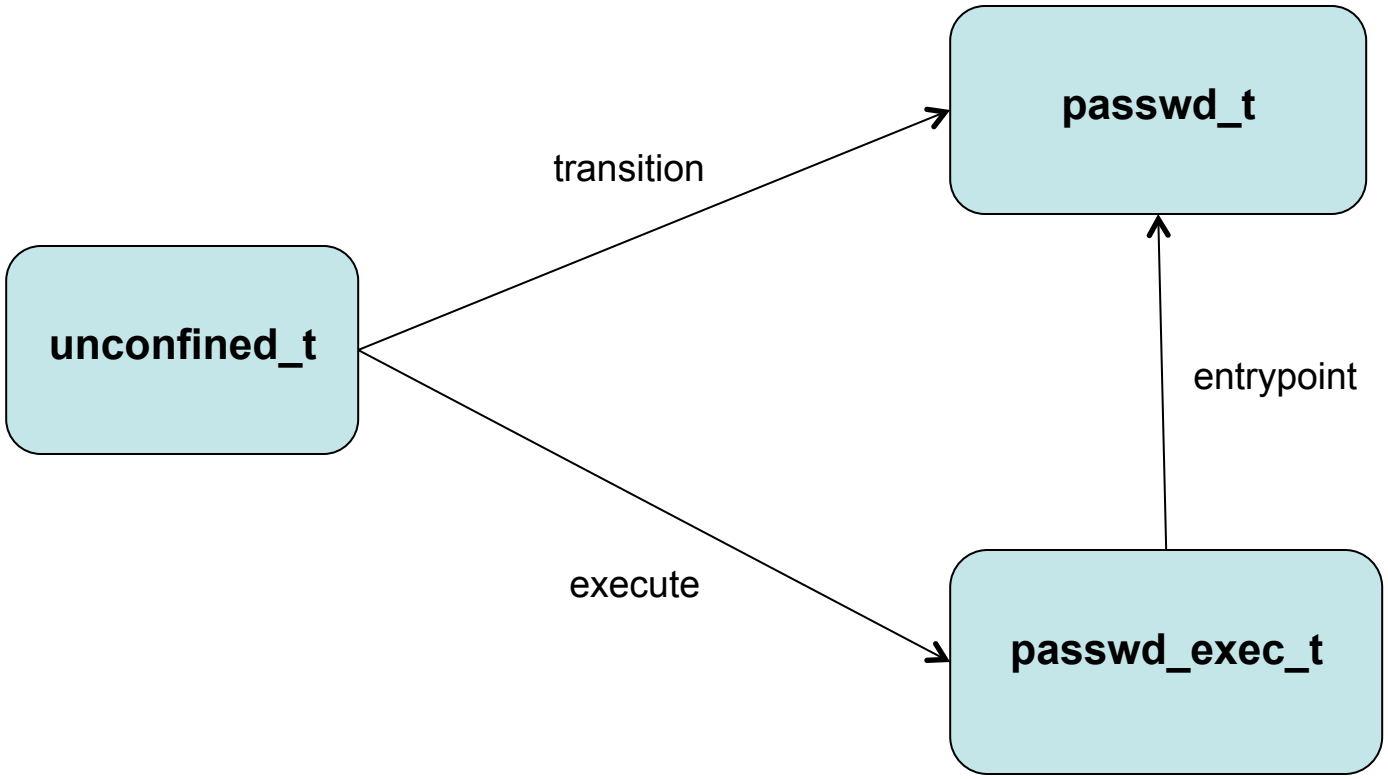
SELinux protects specific operations by allowing them only through type transitions.

The passwd command edits the /etc/shadow file, on behalf of a user, but we don't want the web server to run that command.

In SELinux, we handle this with types:

- passwd_t : the context which is allowed to edit the shadow file
- shadow_t : the shadow file
- passwd_exec_t : the passwd program

Type Transitions Diagram



Ex: Type Transition Rules

Here's the automated transition:

```
# sesearch -T  -t passwd_exec_t
type_transition unconfined_t passwd_exec_t : process passwd_t;
```

We'll need an allow rule for the domain-to-domain transition:

```
# sesearch -A -s unconfined_t -t passwd_t -c process -p transition
allow unconfined_t passwd_t : process transition ;
```

We'll also need an allow rule for `unconfined_t` to run `passwd_exec_t`'s program.

```
# sesearch -A -s unconfined_t -t passwd_exec_t -c file -p execute
allow unconfined_t passwd_exec_t : file { read getattr execute open } ;
```

The `passwd_exec_t` domain will need to be a defined entrypoint for `passwd_t`.

```
# sesearch -A  -s passwd_t -t passwd_exec_t -c file -p entrypoint
allow passwd_t passwd_exec_t : file { … entrypoint …} ;
```

More on Type Transitions

There are more than 12,000 automatic type transitions in the RHEL7 policy.

```
# sesearch -T  | grep process | wc -l
12309
```

Basic Type Enforcement

Let's look at the types applied to the BIND DNS server's files.

```
# ls -lZ /etc/named.conf /var/named/
-rw-r-----. root named system_u:object_r:named_conf_t:s0 named.conf
drwxrwx---. named named system_u:object_r:named_cache_t:s0 data
drwxrwx---. named named system_u:object_r:named_cache_t:s0 dynamic
-rw-r-----. root  named system_u:object_r:named_conf_t:s0 named.ca
-rw-r-----. root  named system_u:object_r:named_zone_t:s0 named.empty
-rw-r-----. root  named system_u:object_r:named_zone_t:s0
named.localhost
-rw-r-----. root  named system_u:object_r:named_zone_t:s0
named.loopback
drwxrwx---. named named system_u:object_r:named_cache_t:s0 slaves
```

named's Type

Let's find out what type `named` runs as.

```
# ps -efZ | grep named
system_u:system_r:named_t:s0    named     61256       1  0 19:37 ?
00:00:00 /usr/sbin/named -u named

# ls -lZ /usr/sbin/named
-rwxr-xr-x. root root system_u:object_r:named_exec_t:s0 /usr/sbin/named
```

The `named_t` type can read the configuration file and write to the zone files.

Experiment with Types

Here are the allow rules that lets named (named_t) access its configuration file (named_conf_t):

```
# sesearch --allow -s named_t -t named_conf_t
Found 4 semantic av rules:
   allow named_t file_type : filesystem getattr ;
   allow named_t named_conf_t : file { ioctl read getattr lock open } ;
   allow named_t named_conf_t : dir { ioctl read getattr lock search
open } ;
   allow named_t named_conf_t : lnk_file { read getattr } ;
```

Let's try changing the type on the named configuration file.

```
# chcon –t admin_home_t /etc/named.conf
```

Starting named

Let's try starting named.

```
# service named start
Redirecting to /bin/systemctl start  named.service
Job for named.service failed. See 'systemctl status named.service' and
'journalctl -xn' for details.
# journalctl -xn
… : SELinux is preventing /usr/sbin/named-checkconf from read access on the file
named.conf.
```

Now, let's change the context back and re-try starting named.

```
# chcon -t named_conf_t /etc/named.conf
# service named start
# ps -ef | grep /usr/sbin/name[d]
named     62700      1  0 20:57 ?        00:00:00 /usr/sbin/named -u named
```

Policy Modules

The SELinux policy is made up of modules.

```
# semodule -l | head -5
abrt    1.4.1
accountsd       1.1.0
acct    1.6.0
afs     1.9.0
aiccu   1.1.0
# semodule -l | wc -l
393
# ls -l /etc/selinux/targeted/modules/active/modules | wc -l
394
```

Making a New Policy Module

Remember how we saw that log message about audit2allow? We can use this tool to create a new policy module and load it.

```
#  grep named-checkconf /var/log/audit/audit.log | audit2allow -M
named-admin-home
# cat named-admin-home.te
…
allow named_t admin_home_t:file read;

# semodule -i named-admin-home.pp
# semodule -l | wc -l
394
```

Labeling New Directories

We could label a new directory with chcon, but if we want it to be part of the policy long-term, we'll need semanage.

```
semanage fcontext -at named_zone_t "/etc/named/zones"
semanage fcontext -at named_zone_t "/etc/named/zones/.*"
```

To then test this and label all the files in /etc/named/zones, we can run restorecon:

```
restorecon –r /etc/named/zones
```

SELinux Booleans

The SELinux targeted policy incorporates boolean variables that serve as on-off switches.

```
# semanage boolean –l
# semanage boolean -l | grep named
named_write_master_zones (off  ,  off)  Determine whether Bind can write to master zone files. Generally this is used for dynamic DNS or zone transfers.
named_tcp_bind_http_port (off  ,  off)  Determine whether Bind can bind tcp socket to http ports.
```

We're going to need to change the first one to allow BIND to sign zones.

Toggling SELinux Booleans

We can set the boolean non-persistently:

```
# getsebool named_write_master_zones
named_write_master_zones --> off
# setsebool named_write_master_zones on
# getsebool named_write_master_zones
named_write_master_zones --> on
```

Once we're sure, let's set it persistently:

```
# setsebool -P named_write_master_zones on
```

This re-compiles the monolithic policy file again.

Changing Ports

SELinux governs ports:

allow named_t dns_port_t : udp_socket { recv_msg send_msg name_bind } ;

To add port 54 to named:

```
# semanage port -a -t dns_port_t -p udp 54
# semanage port -l | grep dns_port_t
dns_port_t                     tcp      53
dns_port_t                     udp      54, 53
```

Giving up a little

SELinux also lets us off the hook sometimes, with permissive domains.

```
semanage permissive -a named_t
```

This isn't simply a gift. It helps avoid the all-or-nothing decision that many sysadmins have made, putting SELinux into permissive mode for life.

SELinux Logging

SELinux logs to `/var/log/audit/audit.log` via `auditd`.

If auditd isn't running, it logs to `/var/log/messages` instead.

If setroubleshootd is running, it logs to both files.

When an access prevention happens, setroubleshootd logs a line that tells us how to get more data:

```
Jul 26 10:13:57 localhost setroubleshoot: SELinux is preventing /usr/sbin/named-
checkconf from read access on the file named.conf. For complete SELinux messages.
run sealert -l eb85bdac-2563-4f73-9a02-ced40ad2d81b
Jul 26 10:13:57 localhost python: SELinux is preventing /usr/sbin/named-checkconf
from read access on the file named.conf.
```

sealert Output

```
SELinux is preventing /usr/sbin/named-checkconf from read access on the file
named.conf.
*****  Plugin catchall (100. confidence) suggests   **************************
If you believe that named-checkconf should be allowed read access on the
named.conf file by default.
…
Do allow this access for now by executing:
# grep named-checkconf /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp
Additional Information:
Source Context                system_u:system_r:named_t:s0
Target Context                system_u:object_r:admin_home_t:s0
Target Objects                named.conf [ file ]
…
Raw Audit Messages
…
```

SELinux Alternatives

SELinux is not the only popular MAC system for Linux.

- AppArmor
- Tomoyo
- SMACK

SELinux Packages to Install

The RHEL7 system we're using for this has the following optional packages installed:

```
policycoreutils-python
policycoreutils-gui
setools-console
setools-gui
setroubleshoot
setroubleshoot-server
```