

S E L I N U X        I N        A C T I O N

Report Prepared by D.Snezhkov (dxs@hush.com)  
for SE573 project

June, 5th, 2005

Table of Contents:.....	1
Introduction and overview.....	2
Pre-requisites.....	3
PHASE I.....	4
PHASE II.....	5
PHASE III.....	8
PHASE IV.....	9
Conclusion.....	11
Bibliography.....	12

-- Introduction and Overview --

Security Enhanced Linux (SELinux) is an extension to the standard Linux kernel that has been designed to enforce strict access controls. SELinux lets you confine processes to the minimum amount of privilege they require. In this report, I will cover the ideas behind SELinux and show how to configure and manage an SELinux system for transparently securing existing applications as well as new ones.

As an example of configuring a security policy, I'll show how to configure out-of-the-box and custom-written ping-like applications with an example security policy that restricts certain users to accessing only the functionality which is required for operation. Also, I will show how SELinux solves the problem of rogue superuser, and prevents privilege elevation. Another goal of this report is to describe the process of writing SELinux-aware applications, as well as give an example of such attempts.

In this report, SELinux policies are configured manually without automated policy-generation tools. I feel such step-by-step walk-through helps to achieve better understanding of the structure, philosophy and inner workings of SELinux product.

There is a definite attempt to present the findings as a 'howto' document as a structure of the report, and situations presented here are attempted to reflect real-life circumstances that SELinux Administrator may encounter. These situations are presented as SELinux configuration phases in the report.

Phase Goals :

PHASE I : Enabling selective functionality in SELinux environment. ( packaged ping application )  
 PHASE II : Properly configuring access permissions for new applications under SELinux ( custom ping-like application ).  
 PHASE III: Restricting rogue superuser and insider from cooperating to circumvent SELinux security  
 PHASE IV : Writing new SELinux aware programs

In addition the policy configuration describes the following :  
 show what root and regular insider can/cannot accomplish within SELinux:  
 show SELinux protection features for SETGID/SETUID commands, network sockets manipulation and access,  
 /proc and /selinux filesystems, access to privileged directories and devices

Participants: 4 Unix/Selinux users

User1 : spike ( legitimate NOC operator )  
 User2 : setest ( non-NOC operator, legitimate user )  
 User3 : root ( as : administrator ( rogue/confused superuser ) )  
 User4 : root ( as : selinux administrator )

Note: for the sake of simplicity both User3 and User4 are represented as 'root' system user.  
 spike and setest are non-privileged users that are not allowed to run privileged commands

Expectations: spike is a non-privileged user that is allowed to:  
 a. run packaged /bin/ping command  
 b. convert privilege from (a) to run custom created /home/spike/my ping command as a separate application to monitor PIX firewall availability  
 c. write a selinux-aware application  
 setest is non-privileged user that is not allowed to  
 a. run packaged /bin/ping command  
 b. run custom create /home/setest/my ping command in ~setest home directory.  
 root is privileged user which is not allowed to modify SELinux policy and not allowed to elevate given privileges to help insider setest to compromise the system

```
--      Pre-requisites      --
```

OS pre-requisites :

```
Users accounts: spike and setest
Admin accounts: root
                Note: root will also play role of selinux administrator in these
                tests.
User directories : /home/spike and /home/setest
Packaged /bin/ping
                (see later)Custom created myping.c which is a part of myapp SELinux
                security environment
```

Note: Create OS users :

```
useradd -c "Spike user" -m -k -d /home/spike -o 1001 spike
useradd -c "SEtest user to test the limits of SELinux " -m -k -d
                /home/setest -o 1000 setest
```

User directories

```
drwx-----  5 spike users 4096 May 26 14:27 /home/spike
drwx-----  4 setest users 4096 May 26 14:32 /home/setest/
```

Check for packaged ping:

```
-rwsr-xr-x  1 root root 32908 Feb 16 2004 /bin/ping
```

SELinux pre-requisites :

User accounts created :

spike and setest are in different roles and domains ( second and third )

in `$SELINUX/users` add :

```
user setest roles { second_r };
user spike roles { third_r };
```

in `$SELINUX/domains/user.te`

```
full_user_role(second)
full_user_role(third)
```

in `$SELINUX/macros/user_macros.te` add :

```
undefine(`in_user_role`)
define(`in_user_role', '
role user_r types $1;
role staff_r types $1;
role second_r types $1; # enable domain transition
role third_r types $1; # enable domain transition
```

As SELinux Administrator ( root in this case ) set proper permissions

on user's home directories :

```
find /home/spike -print0|xargs -0 chcon -h spike:object_r:third_home_t ;
chcon -h spike:object_r:third_home_dir_t /home/spike
find /home/setest -print0|xargs -0 chcon -h setest:object_r:second_home_t ;
chcon -h setest:object_r:second_home_dir_t /home/setest
```

*# Disable enforce mode*

```
echo 0 > /selinux/enforce
```

NOTE:

We use `$SELINUX` variable throughout the tests. This is a shortcut to `/etc/security/selinux/policy/src` directory under Fedora FC3 distribution.

```
--      PHASE I      --
```

In a non-selinux environment non-privileged users are allowed to run `'/bin/ping'` as long as the SUID bit is set in unix permissions. By default, under SELinux such functionality is restricted to users in the privileged role. Our first task is to understand how suh functionality is restricted and only permit user spike to run the utility.

Suppose both spike and setest can ping the pix firewall under permissive mode :

```
[setest@fedora setest]$ ping pix
PING pix (192.168.1.1) 56(84) bytes of data.
64 bytes from pix (192.168.1.1): icmp_seq=0 ttl=255 time=0.620 ms
64 bytes from pix (192.168.1.1): icmp_seq=1 ttl=255 time=0.530 ms

--- pix ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.530/0.575/0.620/0.045 ms, pipe 2
```

```
[spike@fedora spike]$ ping pix
PING pix (192.168.1.1) 56(84) bytes of data.
64 bytes from pix (192.168.1.1): icmp_seq=0 ttl=255 time=0.569 ms
64 bytes from pix (192.168.1.1): icmp_seq=1 ttl=255 time=0.547 ms

--- pix ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.547/0.558/0.569/0.011 ms, pipe 2
```

```
# Enforce !
[root@fedora policy]# echo 1 > /selinux/enforce
```

What happens now ....

```
[setest@fedora setest]$ ping pix
ping: icmp open socket: Permission denied
```

```
[spike@fedora spike]$ ping pix
ping: icmp open socket: Permission denied
```

We only worry about spike who needs to have this functionality enabled. First, let's look at the log and in the log we read :

```
May 29 11:16:52 fedora kernel: audit(1117383412.554:0): avc: denied {
create } for pid=19640 exe=/bin/ping scontext=setest:second_r:second_t
tcontext=setest:second_r:second_t tclass=rawip_socket
May 29 11:16:52 fedora kernel: audit(1117383412.554:0): avc: denied {
net_raw } for pid=19640 exe=/bin/ping capability=13
scontext=setest:second_r:second_t tcontext=setest:second_r:second_t
tclass=capability
May 29 11:16:52 fedora kernel: audit(1117383412.554:0): avc: denied {
setuid } for pid=19640 exe=/bin/ping capability=7
scontext=setest:second_r:second_t tcontext=setest:second_r:second_t
tclass=capability
May 29 11:16:52 fedora kernel: audit(1117383412.556:0): avc: denied {
setopt } for pid=19640 exe=/bin/ping lport=1
scontext=setest:second_r:second_t tcontext=setest:second_r:second_t
tclass=rawip_socket
May 29 11:16:52 fedora kernel: audit(1117383412.556:0): avc: denied {
getopt } for pid=19640 exe=/bin/ping lport=1
scontext=setest:second_r:second_t tcontext=setest:second_r:second_t
tclass=rawip_socket
May 29 11:16:52 fedora kernel: audit(1117383412.556:0): avc: denied {
write } for pid=19640 exe=/bin/ping lport=1
scontext=setest:second_r:second_t tcontext=setest:second_r:second_t
tclass=rawip_socket
May 29 11:16:52 fedora kernel: audit(1117383412.557:0): avc: denied {
read } for pid=19640 exe=/bin/ping lport=1
scontext=setest:second_r:second_t tcontext=setest:second_r:second_t
tclass=rawip_socket
```

These are the system calls and groups of macros that second\_r:second\_t context cannot execute on rawip\_socket target security class of objects, as well as enter certain linux capabilities such as setuid.

So, we analyze the access pattern and add the following in `$SELINUX/domain/misc/custom.te` :

```
allow third_t self:capability { setuid net_raw };
allow third_t third_t:rawip_socket { create_socket_perms };
```

Of course we rebuild and load our policy:

```
in $SELINUX :
    make clean; make install
```

Spike is now able to use `'/bin/ping'` since all the dependencies are resolved.

```
[spike@fedora spike]$ ping pix
PING pix (192.168.1.1) 56(84) bytes of data.
64 bytes from pix (192.168.1.1): icmp_seq=0 ttl=255 time=0.567 ms
```

```
--- pix ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.567/0.567/0.567/0.000 ms, pipe 2
```

```
... Unlike setest
[setest@fedora setest]$ ping pix
ping: icmp open socket: Permission denied
```

We can also take a look under the hood to uncover how setest is not able to get access to all the resources.

```
# Setest can see why she cannot ping pix : SOCK_RAW cannot be opened!
[setest@fedora setest]$ strace -qenetwork ping pix
```

```
-----snip-----
socket(PF_INET, SOCK_RAW, IPPROTO_ICMP) = -1 EACCES (Permission denied)
socket(PF_FILE, SOCK_STREAM, 0) = 3
connect(3, {sa_family=AF_FILE, path="/var/run/nscd/socket"}, 110) = -1
ENOENT (No such file or directory)
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(1025),
sin_addr=inet_addr("192.168.11")}, 16) = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(32800),
sin_addr=inet_addr("192.168.1.10")}, [16]) = 0
ping: icmp open socket: Permission denied
-----snip-----
```

These are the steps neede to ba taken when analyzing how to secure a given application under SELinux and design appropriate policy if necessary.

-- PHASE II --

Ok, so we have allowed spike to run a ping.  
But Suppose, spike is legitimate user that, say , is an operator in NOC and truly needs to have the ping functionality to check on availability of nodes. However, if we allow third\_r role to enter third\_t domain which can manipulate raw sockets, it will be exploited sooner or later. Such functionality should be confined to a few controlled applications. Even though spike may write it, the application needs to pass a security audit.

And what happens if spike wants to write his own ping program and execute it without security team;s blessing. Can;t he just do it ?  
Under normal circumstances ( no SELinux enforce ) he can do that.

```
Can he circumvent SeLinux perms :
[spike@fedora spike]$ gcc -o myping.bin myping.c
-rwxr-xr-x spike users spike:object_r:third_home_t myping.bin
```

```
Spike looks up permissions under ping and wants to simulate ping domain
[spike@fedora spike]$ ls -Z /bin/ping
-rwsr-xr-x+ root root system_u:object_r:ping_exec_t /bin/ping
```

```
[spike@fedora spike]$ chcon -h spike:object_r:ping_exec_t myping
chcon: failed to change context of myping to spike:object_r:ping_exec_t:
Permission denied
```

We can see why he is not successful : there is no rule to enable spike to label his files to ping\_exec\_t domain.

```
[root@fedora policy]# setfiles -d -v file_contexts/file_contexts
# /home/spike/(*)
setfiles: read 1451 specifications
setfiles: labeling files under /home/spike/myapp
setfiles: /home/spike/myping matched by
```

```

                (/home/spike/.,spike:object_r:third_home_t)
setfiles: hash table stats: 1 elements, 1/65536 buckets used, longest chain
length 1
setfiles: labeling files under /home/spike/myping.c
setfiles: /home/spike/myping.c matched by
(/home/spike/.,spike:object_r:third_home_t)
setfiles: hash table stats: 1 elements, 1/65536 buckets used, longest chain
length 1
setfiles: Done.

```

We can see that in the log ... ( notification ? )

```

May 29 11:35:13 fedora kernel: audit(1117384513.053:0): avc: denied {
relabelto } for pid=19754 exe=/usr/bin/chcon name=myping dev=hdcl ino=795096
scontext=spike:third_r:third_t tcontext=spike:object_r:ping_exec_t tclass=file

```

Also, notice that /bin/ping is SUID to root which spike may not able to set. He needs some help ...

Ok, so spike has failed executing his program and sends a request to root to make his myping.bin SUID so he can perform his tasks.

The system administrator is not entirely familiar with SELinux policies and configuration, The only thing he knows is what he is told by the Security Team : to change his role from staff\_r to sysadm\_r using newrole -r command when something is not working...

So, his solution to spike's problem is to check for basic security structure of myping.c

and upon approval do the following :

```

[root@fedora policy]# chown root:root /home/spike/myping.bin
[root@fedora policy]# chmod u+s /home/spike/myping.bin
[root@fedora policy]# ls -lZ /home/spike/myping.bin

```

```

-rwsr-xr-x+ root      root      spike:object_r:third_home_t
/home/spike/myping.bin

```

This works for spike :

```

[spike@fedora spike]$ ./myping.bin pix
PING pix (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 time=0 ms
64 bytes from 192.168.1.1: icmp_seq=1 time=0 ms

```

... until the security audit time comes and the SELinux Policy is modified to restrict unprivileged users from direct manipulation on raw sockets. Essentially, custom.te ( see pre-requisites above ) template was removed to disallow setuid, create\_socket and net\_ipraw and others ....

```

( remember ? :
  in $SELINUX/domain/misc/custom.te :
allow third_t self:capability { setuid net_raw };
allow third_t third_t:rawip_socket { create_socket_perms };
)

```

So, next time spike runs his application he sees :

```

[spike@fedora spike]$ ./myping.bin pix
ping: socket: Operation not permitted

```

The request to Properly configure custom written ping application is sent to the Security Team, as it should have been done from the start...

After carefully reviewing the code and approval, SELinux Administrator needs to encapsulate myping.bin into a policy. She creates a separate domain (myapp\_exec\_t) which myping process may enter :

```

1. in $SELINUX/file_contexts/misc/myapp.fc add:
#myapp

```

```
/home/spike/myping -- spike:object_r:myapp_exec_t
```

2. Create `$SELLINUX/domains/misc/myapp.te`

This would look something like this :

```
----- start myapp.te -----
# myapp_t is the domain for the myping.bin program.
# myapp_exec_t is the type of the corresponding program.
#
type myapp_t, domain, privlog; # can log
role system_r types myapp_t; # of course, allow superuser access
role third_r types myapp_t; # allow third_t and consequently spike user
# totransition

in_user_role(myapp_t) # declare context transition of third_r into
# type/domain myapp_t
type myapp_exec_t, file_type, sysadmfile, exec_type;

# Transition into this domain when you run this program.
domain_auto_trans(sysadm_t, myapp_exec_t, myapp_t) # sysadmin is allowed
domain_auto_trans(initrc_t, myapp_exec_t, myapp_t) # allow init to
# transition to myapp_t on boot

domain_auto_trans(third_t, myapp_exec_t, myapp_t) # third_t can
# transition to myapp_t while running myping

uses_shlib(myapp_t) # network libraries, resolver, etc
can_network(myapp_t)
can_yplib(myapp_t) # For lookups via /etc/nsswitch.conf
allow myapp_t etc_t:file { getattr read }; # This is for /etc/services and
# the like

allow myapp_t self:unix_stream_socket create_socket_perms; # UNIX domain
# sockets, pipes

# Let myapp create raw ICMP packets.
allow myapp_t self:rawip_socket { create ioctl read write bind getopt setopt };
# Needed for socket manipulation

allow myapp_t netif_type:netif { rawip_send rawip_recv }; # Is needed for ICMP
allow myapp_t node_type:node { rawip_send rawip_recv }; # Is needed for ICMP

# Use capabilities.
allow myapp_t self:capability { net_raw setuid }; # This is familiar (
# recall /bin/ping enablement )

# Access the terminal.
allow myapp_t admin_tty_type:chr_file rw_file_perms; # Only when running
# as admin not user

dontaudit myapp_t fs_t:filesystem getattr; # errors when going to
# read some irrelevant fs info

# it tries to access /var/run
dontaudit myapp_t var_t:dir search; # for socket and pipes ...

----- end myapp.te -----
```

3. build policy :

```
make clean
make load
/usr/sbin/load_policy /etc/security/selinux/policy.`cat
/selinux/policyvers`
touch tmp/load
make install
```

4. `chcon -h spike:object_r:myapp_exec_t /home/spike/myping # In case`  
`# the file_contents/file_contents relabel REGEXP matched third_home_t`  
`# type instead of myapp_exec_t`

```
[spike@fedora spike]$ ls -Z
-rwxr-xr-x spike users spike:object_r:myapp_exec_t myping.bin
```

```
SELinux Administrator tests the configuration :
[spike@fedora spike]$ ./myping.bin pix
-- HANG --
```

```
-- HANG --
-- HANG --
-- HANG --
-- HANG --
```

Uh-oh, there;s a problem somewhere ...  
Ok, what do we see in the log :

```
May 29 12:40:34 fedora kernel: audit(1117388434.793:0): avc: denied {
write } for pid=21426 exe=/home/spike/myping.bin path=/ dev=hdcl ino=135699
scontext=spike:third_r:myapp_t tcontext=system_u:object_r:root_t
tclass=chr_file
```

Looks like the application is denied write to some character file.What would that be ?

```
[root@fedora policy]# df | grep hdcl
/dev/hdcl          10079324    3064240    6503072    33% /
find / -inum 135699      yields /dev/pts/5
```

Oh, it worked, but it simply could not write to the terminal :

Add the following line into the myapps.te :

```
allow myapp_t { ttyfile ptyfile }:chr_file rw_file_perms;
```

Repeat the test :

```
[spike@fedora spike]$ ./myping.bin pix
PING pix (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 time=0 ms
64 bytes from 192.168.1.1: icmp_seq=1 time=0 ms
64 bytes from 192.168.1.1: icmp_seq=2 time=0 ms
```

Nice! We've got a working SELinux policy for a custom application.

-- PHASE III --

What about setest ? Can she execute the same ?

Consider the scenario when setest got the source code for myping.c and also she is aware of the environment she will run the application in, that is, SELinux. Or, perhaps, she tricked ( social engineering ? ) the same superuser that worked with spike earlier to give her access to myping utility.

Consider :

1. Root trusts selinux and does the foolowing for her :

```
[root@fedora root]# cp /home/spike/myping.bin /home/setest/myping.bin
cp: accessing "/home/setest/myping.bin": Permission denied
????
[root@fedora policy]# id
id=0(root)
gid=0(root)groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
context=root:staff_r:staff_t
```

*#Root remembers what he needs to do to fix this ...*

```
[root@fedora policy]# newrole -r sysadm_r
Authenticating root.
Password:
[root@fedora policy]# cp /home/spike/myping.bin
/home/setest/myping.bin
cp: overwrite "/home/setest/myping.bin"? y
[root@fedora policy]# ls -l /home/setest/myping.bin
-rwsr-xr-x 1 root root 15235 May 29 12:10 /home/setest/myping.bin
[root@fedora policy]#
```

*#Setest tries to execute the application :*

```
[setest@fedora setest]$ ./myping.bin pix
```

```
ping: socket: Permission denied
May 29 13:10:04 fedora kernel: audit(1117390204.382:0): avc: denied {
create } for pid=21613 exe=/home/setest/myping.bin
scontext=setest:second_r:second_t tcontext=setest:second_r:second_t
tclass=rawip_socket
```

Fails ...

2. Root ponders a bit, and checks the following since he already seen this error before recovering:

```
[root@fedora policy]# ls -lZ /home/setest/myping.bin
-rwsr-xr-x+ root root setest:object_r:second_home_t
/home/setest/myping.bin
```

As we can see the root user which does not have access to selinux - enabling role cannot make working copies of the SUID file. From the surface everything seems fine to root, but context does not match. Ok, now the root has learned the lessons from the audit which disabled spike's ability to run myping application. Now, root learns of the following command :

```
chcon -h setest:object_r:myapp_exec_t /home/setest/myping.bin
```

( FYI: the following would have been shown in the log if root ran previous command as staff\_r:staff\_t role, so this could be restricted )

In the log we see :

```
May 29 12:42:52 fedora kernel: audit(1117388572.066:0): avc: denied {
dac_read_search } for pid=21511 exe=/usr/bin/chcon capability=2
scontext=root:staff_r:staff_t tcontext=root:staff_r:staff_t tclass=capability
-- OR --
May 29 14:24:04 fedora kernel: audit(1117394644.151:0): avc: denied {
getattr } for pid=21750 exe=/bin/bash path=/etc/security/selinux/src
dev=hdcl ino=748517scontext=root:staff_r:staff_t
tcontext=system_u:object_r:policy_src_t tclass=dir
May 29 14:24:04 fedora kernel: audit(1117394644.151:0): avc: denied {
search } for pid=21750 exe=/bin/bash name=src dev=hdcl ino=748517
scontext=root:staff_r:staff_t tcontext=system_u:object_r:policy_src_t
tclass=dir
```

The attempt fails this time, however, suppose, such functionality is granted to root ...

So, in the end the myping.bin in bot ~setest and ~spike are identical !

```
-rwsr-xr-x root root spike:object_r:myapp_exec_t myping.bin
-rwsr-xr-x root root setest:object_r:myapp_exec_t myping.bin
```

Yep, root can do this. Does that mean we cannot control who runs what when the superuser is involved ... ?  
Not really. As long as proper permissions and policies are set for myapp type, and root is restricted from modifying and reloading policies ( or subset ) everything should be fine.

But watch: as the following shows ...

3. Setest tries one more time.  
[setest@fedora setest]\$ ./myping.bin pix  
ping: socket: Permission denied

Nope, she cannot do it even though the selinux context and permissions are identical to spike :

This situation shows how Selinux can protect the environment from insider attacks and it also enables superuser RBAC, which downgrades the superuser to an almost regular user with granted and denied functionality.

-- PHASE IV --

How can we incorporate some SELinux functionality into an application

itself programmatically?

Take the same myping.c application. Wouldn't it be nice to know what context the application is running under, and maybe, branch out inside depending on the context.

Consider the following function which can be inserted into myping.c:

```

----- Start get_se_attribute-----

int get_se_attributes(void)
{
    char *buf;
    char mode[4];

    int rc;
    int enforce;
    int enabled;

    if ( (enabled=is_selinux_enabled()) ==0 ){
        fprintf(stderr, "SELinux is not enabled\n");
    }else {

enforce=security_getenforce();
        switch (enforce) {
            case 1:
                snprintf(mode,4,"ENF"); // Enforcing
                break;
            case 0:
                snprintf(mode,4,"PRM"); // Permissive
                break;
            case -1:
                snprintf(mode,4,"DIS"); // Disabled
                break;
        }
        printf("OK, running under SELinux [ mode = %s ] as : ", mode);

        rc = getcon(&buf);

        if (rc < 0) {
            fprintf(stderr, "getcon() failed\n");
            exit(2);
        }
        printf("%s\n", buf);
        freecon(buf);

    }
    return(0);
}

----- End get_se_attributes -----

```

Inserted at the beginning of myping.c:

```

/* SELinux functionality */
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <selinux/selinux.h>
#include <selinux/get_default_type.h>

```

```

int get_se_attributes(void);
/* SELinux functionality */

```

```

<code before the function>
get_se_attributes();
<code after the function>

```

Spike compiles the application

```
[spike@fedora spike]$ gcc -o myping.bin -L/usr/lib -lselinux myping.c
```

Note : Of course, spike loses all permissions and myapp\_t domain on the file that he creates :

```
-rwxr-xr-x spike users spike:object_r:third_home_t myping.bin
```

On the first run we get errors :

```
[spike@fedora spike]$ ./myping.bin pix
OK, running under SELinux [ mode =
getcon() failed
```

The log shows :

```
May 29 17:21:50 fedora kernel: audit(1117405310.052:0): avc: denied {
search } for pid=22663 exe=/home/spike/myping.bin dev=proc ino=1
scontext=spike:third_r:myapp_t tcontext=system_u:object_r:proc_t
tclass=dir
May 29 17:21:50 fedora kernel: audit(1117405310.053:0): avc: denied {
search } for pid=22663 exe=/home/spike/myping.bin dev=proc ino=1
scontext=spike:third_r:myapp_t tcontext=system_u:object_r:proc_t tclass=dir
May 29 17:21:50 fedora kernel: audit(1117405310.053:0): avc: denied {
search } for pid=22663 exe=/home/spike/myping.bin dev=proc ino=1
scontext=spike:third_r:myapp_t tcontext=system_u:object_r:proc_t tclass=dir
May 29 18:01:04 fedora kernel: audit(1117407664.890:0): avc: denied {
getattr } for pid=23509 exe=/home/spike/myping.bin path=/proc/23509/mounts
dev=proc ino=1540685840 scontext=spike:third_r:myapp_t
tcontext=spike:third_r:myapp_t tclass=file
May 29 18:01:04 fedora kernel: audit(1117407664.891:0): avc: denied {
search } for pid=23509 exe=/home/spike/myping.bin dev=selinuxfs ino=1017
scontext=spike:third_r:myapp_t tcontext=system_u:object_r:security_t tclass=dir
```

We need to add the following lines into `$SELINUX/domains/misc/myapp.te` :

```
allow myapp_t { self proc_t }:dir { search };           #enable /proc/<pid>*
#read search
allow myapp_t { self proc_t }:lnk_file { read };
allow myapp_t { self proc_t }:file { read getattr };   # enable links
#in /proc to be read and stat()
```

Who am I ?

```
[spike@fedora spike]$ id uid=1001(spike) gid=100(users) groups=100(users)
context=spike:third_r:third_t
```

Who am I now ?

```
[spike@fedora spike]$ ./myping.bin pix
OK, running under SELinux [ mode = PRM ] as : spike:third_r:myapp_t
PING pix (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 time=6 ms
64 bytes from 192.168.1.1: icmp_seq=1 time=0 ms
```

As we can see not only we were able to see how `spike;s` application entered the `myapp_t` domain but also were able to get basic information about the SELinux environment the process in `myapp_t` was allowed to gather. Given this ability the applications can enforce/coordinate their internal security mechanisms with SELinux policy.

-- Conclusion --

## -- Bibliography --

1. SELinux Technology from NSA <http://www.nsa.gov/selinux/>
2. SELinux Symposium <http://www.selinux-symposium.org/>
3. SELinux at MITRE <http://www.mitre.org/tech/selinux/>
4. Fedora SELinux project <http://fedora.redhat.com/projects/selinux/>
5. Hardened Gentoo's SELinuxProject  
<http://www.gentoo.org/proj/en/hardened/selinux/index.xml>
6. Russell Coker's SELinux Site <http://www.coker.com.au/selinux/>
7. Manoj Srivastava's SELinux page  
<http://www.golden-gryphon.com/software/security/selinux.shtml>
8. Carsten's German/English SELinux Wiki Site  
<http://www.securityenhancedlinux.de/>
9. Apache HTTP and SELinux  
<http://fedora.redhat.com/docs/selinux-apache-fc3/>
  
10. <http://www.oreilly.com/catalog/selinux/>
11. <http://www.crypt.gen.nz/selinux/faq.html>
12. <http://infosecuritymag.techtarget.com/2002/mar/techtalk.shtml>
13. <http://www-106.ibm.com/developerworks/library/s-selinux/index.html>
14. <http://www-106.ibm.com/developerworks/library/s-selinux2/index.html>
15. [http://www.crypt.gen.nz/selinux/install\\_fedora.html](http://www.crypt.gen.nz/selinux/install_fedora.html)
16. <http://www.tresys.com/Downloads/whitepapers/pol-struct.pdf>

~