Mandatory Access Control & SELinux

Daniel Estor Muteer Arshad
Email: {danes231,mutar655}@student.liu.se
Supervisor: David Byers, {davby@ida.liu.se}
Project Report for Information Security Course
Linköpings universitetet, Sweden

Abstlract

This report covers SELinux in theory and in practice. It consists of an overview of SELinux and the results of our practical work of developing a customized policy for the ping command. The report confirms the effectiveness of SELinux and shows that by now SELinux has even become fairly easy to configure.

1. Introduction

Mandatory access control (MAC) is known as a strong means for protecting confidentiality and integrity of data. Developed for the use in high-end systems that deal with very sensitive information, MAC was used exclusively in such systems for a long time. In mainstream operating systems the strongest access control was discretionary access control (DAC), like the well-known Unix DAC with users, groups and `rwx` permissions.

With the release of SELinux by the NSA in 2000, MAC is starting to become common even in mainstream operating systems as Linux. Introducing MAC into Linux is clearly a gain of security. However, SELinux is often criticized to be too complex to use. In fact, one must have a very good knowledge about SELinux in order to configure it correctly.

Section 2 provides necessary background information. The theoretical part is completed by section 3, which describes the SELinux policy. The practical part starts with a general description of what you have to do in order to run SELinux. It is followed by a section about our project of developing a customized security policy for the ping command on a given system. Finally section 6 gives our conclusions about SELinux and this project.

2. Background

In this section we give some background information that can be useful for understanding SELinux. We use the terms subject and object as they are commonly used in the context of computer security. Thus we refer to subjects as entities that can perform actions, which are usually processes and to objects as entities that access is performed on like for example files, directories or `network_sockets`.

2.1 Multilevel Security

Every organization and business wishes to secure its information especially such information whose leakage results in greater loss. However they are not willing to lose any of their information but they tolerate the leakage of little and some non critical information. This issue becomes more significant for the defense community, where the loss of sensitive information always result in huge loss, so such organizations never tolerate even a small information leakage. So as a remedy to this multi level security (MLS) technique is introduced in which both people and information is categorized into different levels of trust and sensitivity. The level of trust that is given to user is termed as “clearance level” and the level of trust given to the information is termed as “classification level”. There are four different types of classification levels, namely: Unclassified, Confidential, Secret and Top Secret. So now in order to access the classified information user is first required to establish his trustworthiness by earning clearance to some security level, for instance if user succeeded to earn Secret clearance then he is entitled to view any of the information with the Secret tag.

2.2 Access Control

Access control refers to the mechanism that is used to permit or deny the right of an individual or application to perform some actions. It works by comparing the identity of an individual or application with an access control database. Access Control systems include File permissions (such as create, read, edit or delete on a file server), Program permissions (such as the right to execute a program on an application server) and Data rights (such as the right to retrieve or update

information in a database). In the following we will discuss various techniques of access control.

2.2.1 Discretionary Access Control

Discretionary access control (DAC) is an access policy determined by the owner of the object. The owner decides whether to grant or reject access to the object along with the privileges.

2.2.2 Mandatory Access Control

Mandatory access control is an access policy established by the operating system. Both subjects and objects are assigned security attributes which are used to determine if a subject may access an object. The rules defining which accesses are allowed are stored in the security policy. This policy cannot be changed by the users and on every access the operating system enforces all policy rules to be fulfilled. This means for example that, unlike in DAC, users cannot grant privileges to other users.

2.2.3 Role Based Access Control

Role Based Access Control (RBAC) is another type of access control. It is also called as role based security. This type of access control is used to restrict system access to authorized users only. In RBAC each user is associated with a certain role and on the basis of those roles all the access decisions are made.

In RBAC each user is granted a membership to certain roles based on operations which he is authorized to perform. The operations that user is permitted to perform is determined by his role. In other words the access policy is defined for each role and on the basis of which a user is allowed or restricted to access a certain resource. RBAC also allowed to reap the benefits of least privileged access i.e. Access policy is written in such a way that the user can be given no more privilege that is necessary to perform the operation for which the user is entitled to perform.

2.3 Why is MAC necessary?

DAC alone is not effective enough in protecting the system from the effects of exploited vulnerabilities in application software. This can be illustrated by the example of the ping command on a Linux operating system. Since ping needs to create raw sockets, which only the superuser root is allowed to do, ping must be run by the user root. But ping is a useful command that also normal users should be able to run. Hence ping is a setuid executable, enabling a normal user to run the ping command, which then effectively runs under the superuser root. If this command contains a vulnerability, an attacker can (for example by a buffer overflow attack) gain access to the system as root.

The standard way to prevent such kinds of attacks has been to try to eliminate vulnerabilities from software, but as long as software is written by people, there will always be bugs or vulnerabilities in software. Mandatory access control with SELinux, in contrast, offers a totally different way of preventing such attacks. If it is properly configured, it can confine a command to the least privilege, so that an exploitation of vulnerability in software can cause no harm on the operating system.

2.4 SELinux Architecture

SELinux is an implementation of the FLASK architecture that has been developed to simplify the application of MAC. The main feature of the FLASK architecture is the separation of the policy enforcement from the access decisions. Access decisions are made according to security attributes of subjects and objects, the so called security contexts. We will explain this security context in more detail in section 3.3. A simple layout of the FLASK architecture is shown in Figure 1.

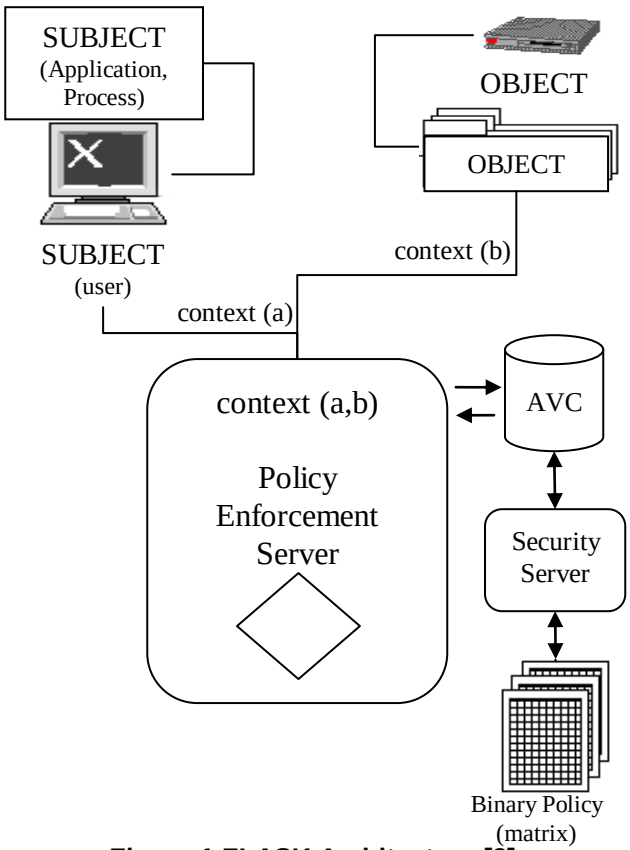

Figure 1 FLASK Architecture [2]

The policy enforcement server catches the access that a subject attempts to perform on an object and gathers their security contexts. These security contexts are passed to the security server that makes the decision. To speed up this process the access vector cache (AVC) is placed between policy enforcement server and security server. After the security server has made the decision, the result is put into the AVC where it can be retrieved by the policy enforcement server. Then the policy server applies this decision by granting or denying the access, whereas in the latter case an avc: denied log message is created.

It should be noted that SELinux does not replace the standard Linux DAC, but is rather placed after the DAC checks. This means that an access that is denied by the DAC is not checked by SELinux and hence no log about this denied access is produced.

3. SELinux Policy

In this section we describe the main concepts of the SELinux policy. The SELinux policy is quite well documented Stephen Smalley's SELinux policy guide [1] and in the Red Hat SELinux user [2], so the information in this section is based on these two sources. In SELinux every subject and object has a security context, which describes its security attributes that are used for access decisions. The most important part of the security context is the type attribute which is used for type enforcement. Closely related to type enforcement is domain transition, which is the change of a subject's type in its security context. Besides, at least two other components of the security context, user and role, are used to make access decisions more fine-grained.

3.1 Security Context

The security context is assigned to every subject and object. It is made up of four components, which are user, role, type and an MLS component. The single components are divided by colons, so an example security context can look like;

```
system_u:object_r:etc_t:s0.
```

In the following sections we will further explain the purpose of the user, role and type component. If MLS is not used, the fourth component is not necessary and is always set to `s0`. Since MLS was not part of the project, we will not consider this component in our further investigations.

As mentioned before, every subject and object has a security context. For files on a file system with extended attributes like ext3 the security context is stored directly in the inode. For new objects or subjects a security context is created according to a global policy determined in different files in the directory `/etc/selinux/default/context/`. For example a new file created in `/etc/` usually gets the security context `system_u:object_r:etc_t:s0`.

3.2 Type Enforcement

The main concept for access decisions in SELinux is type enforcement. In the traditional type enforcement model each subject is assigned a domain and each object is assigned a type. An access is granted if there exists a rule that allows subjects of the given domain to access objects of the given type.

In SELinux the domain of a subject and the type of an object both correspond to the type component of the security context. By convention every type ends in `_t`, like `etc_t`, which is the type of files in `/etc or user_t` which is a commonly used domain in certain SELinux policies. Compared to the standard type enforcement model, the type enforcement mechanism of SELinux is extended. SELinux additionally uses security classes and permissions for those classes that allow a finer access control. The security classes group subjects or objects into certain classes, examples for security classes are file, dir, tcp_socket for objects or process for subjects. Every class is assigned permissions that determine the actions that can be executed on subjects or objects of the respective class. The file class, for example, has permissions like create, read, write or execute, whereas the process class has permissions like transition, which means the process transitions into another domain, or sigchld. Please note that the term permission does not mean that classes are granted these permissions automatically, it rather describes actions that can be permitted by explicit allow rules in the policy. An allow rule always has the following form:

```
allow source_type target_type:class permission;
```

A rule that allows the type passwd_t to read and write files of the type shadow_t looks as follows:

```
allow passwd_t shadow_t:file {read write};
```

One can see here that the two permissions can be grouped together in curly braces instead of writing two different rules. This is also possible for the types and class.

3.3 Domain Transition

When a process is started, it inherits the domain of the process that has created it. In order to be run in its own domain confining the process, a domain transition has to take place. To allow a domain transition three rules are required. These rules become obvious in the following example. A user runs its shell in the domain

user_u. When he executes the passwd command, it should transition to the domain passwd_t. The executable file of the passwd command has the type passwd_exec_t. Then the following three rules allowe the domain transition:

```
allow user_t passwd_exec_t:file {getattr execute};
allow passwd_t passwd_exec_t:file entrypoint;
allow user_t passwd_t:process transition;
```

The first rule allows processes in the domain user_t to execute files of the type passwd_exec_t, which means that the user is allowed to execute passwd in his shell. The second rule specifies an entry point. This means that the domain passwd_t can be entered by executing a file of the type passwd_exec_t. Finally the last rule allows a domain transition from the user_t domain to the passwd_t domain. Note that all three rules are necessary for the domain transition to be allowed.

However, these rules only allow a domain transition; they do not automatically make the process change its domain. For this purpose an explicit type_transition rule is required:

```
type_transition user_t passwd_exec_t:process passwd_t;
```

Since domain transition is very often necessary, there exist, as for many other often used groups of rules, macros that simplify creating domain transition rules. Using these macros the rules for the type transition can be written in the following way:

```
domain_entry_file(passwd_t, passwd_exec_t)
domain_auto_trans(user_t,passwd_exec_t,passwd_t)
```

3.4 Users and Roles

Besides type enforcement, SELinux also uses users and roles to make access decisions. Users can be considered similar to the Linux users, however users in SELinux are not identical to Linux users. Linux users are rather mapped to SELinux users. Common SELinux users are root or user_u. Except for root every SELinux user ends on _u by convention. The purpose of these different users is to further refine the access decisions based on type enforcement. The distinction between different users makes it possible to restrict a domain to special users.

However, the decision, if a user may access a domain is not made directly dependent on the user. For this decision an additional layer, the role is used. Every SELinux user can have different roles and for every domain there must be defined which roles may access it. As for types and users, there also exist naming conventions for roles, which is that all roles end in _r.

4. SELinux in Practice

Here we describe how SELinux can be used and configured in practice. Since we only ran SELinux on Debian lenny during our project, we mainly concentrate on how to deploy SELinux on a Debian system. Sometimes we also refer to fedora because fedora is the distribution with the biggest out of the box support for SELinux.

4.1 Installation and Basic Configuration

In contrast to fedora, SELinux is usually not automatically installed on a Debian system. In order to use SELinux on Debian lenny with the targeted policy, the packages selinux-basics and selinux-policy-default have to be installed. After that selinux-activate needs to be run to configure the boot manager and PAM and to create the file /.autorelabel. After a reboot the file system will be relabeled according to the information in /.autorelabel and after that SELinux should be running in permissive mode. This means access which is not allowed by the policy will not be blocked, but only logged. If forbidden accesses should be blocked SELinux must run in enforcing mode. During runtime the mode can be set via the command setenforce, where setenforce 1 changes to enforcing and schanges to permissive.

The basic configuration is done in the file /etc/selinux/config. There one can determine in which mode SELinux should boot, if it should be enabled and which policy is used.

4.2 Targeted and Strict Policy

The first time after the release of SELinux, configuring the policy was very complex and difficult. Hence effort has been made to simplify this by developing the reference policy. Due to its modular design the reference policy has simplified policy configuration considerably. Today it is available in two versions, the targeted and strict policy.

The special feature of the targeted policy compared to the strict policy is the unconfined_t domain. Processes that run in this unconfined_t domain do not have many access restrictions and effectively run as if they would do without SELinux. In fact, most processes run in the unconfined_t domain and by default only some targeted daemons run in their own domains. This really simplifies the work for the administrator, who does not have to configure the SELinux policy for every application he wants to use on the system. However, the administrator is not restricted to having only some targeted daemons confined by running in their own domain. For every application that he can define its own domain, write a policy module for this application and

load this module into the policy. Thus starting from a fully functional system, a policy with all desired confinements can be developed.

In contrast to that the strict policy requires own domains for every process. Since we did not use the strict policy through our project and Linux distributions like Debian lenny or fedora only offer the targeted policy by default, we concentrate on the targeted policy.

4.3 Developing a Customized Policy Module Under the Targeted Policy

Under the targeted policy an application can easily be confined by developing a policy module for this application. At this place we give a step by step guide that describes how a customized policy module can be developed. How this is used in practice will be described in detail in section 5.2.

The first step of policy module development for an application myapp is to create the files `myapp.te` and `myapp.fc`. The myapp.te file should contain the new type definitions, probably these will be `myapp_exec_t` for the executable file and `myapp_t` for the domain, and the appropriate domain transitions. If at this point further necessary rules are already known, they can also be put into `myapp.te`. The `myapp.fc` file is used to describe the file contexts that have to be added to the policy. Every line in the `myapp.fc` must consist of three columns. The first column contains the file that should get a new security context. This is probably the executable file of the application and perhaps additional files. The second column usually contains `--`, which means that all kinds of files are considered in the file context. Finally, the third column contains the desired security context.

With these two files a first version of the policy module can be compiled. On Debian lenny a suitable makefile can be found under:

```
/usr/share/selinux/default/include/Makefile.
```

after executing `make`:

```
-f  /usr/share/selinux/default/include/Makefile,
```

the module can be loaded with

```
make load -f usr/share/selinux/default/include/Makefile
```

For the first use of the module, the security contexts still have to be applied to the files mentioned in `myapp.fc`. This is done with the `restorecon` command. Now SELinux should be run in permissive mode to get logs about possible access denials for the application. These denial messages can be used to add further allow rules to the `myapp.te` file to give the application the needed privileges. The allow rules can either be created manually from the log messages or by using the `audit2allow` tool, which can automatically create allow rules from the logfile. With the updated `.te` file the module can then be recompiled and the application be run with the updated policy. Now there might be new access denials, which can be treated in the same way as before. This whole procedure of running the application, looking at the log files, adding new rules to the .te file and recompiling the policy module should be repeated until the application runs as it is supposed to be. At that point SELinux can be switched into the enforcing mode again.

5. A Customized Policy for ping

This section covers the results of our practical work with SELinux. Our task was to develop a customized policy for a modified ping command on a given system. The policy should effectively prevent successful attacks on this ping command.

5.1 Initial Situation

The system we worked on was Debian lenny run in a UML virtual machine. On this system the modified ping command was installed under the name `pong` with the executable `/bin/pong`. The modification to the ordinary ping command was a vulnerability in then handling of the `-I` option that can be exploited to get a root shell. By running an exploit according to given instructions we could confirm the vulnerability.

The basic SELinux packages were already installed, the only new package we installed was `selinux-policy-default`. We installed this package to replace the deprecated policy packages that Debian had offered until etch.

5.2 Development of the Policy for the pong Command

In developing the policy for pong we proceeded as described previously. We started with the minimal policy that provides a transition of pong to the pong_t domain on running. Our initial pong.te file looked the following:

```
policy_module(pong,1.0.0)
########################################
#
# Declarations
#
type pong_t;
type pong_exec_t;
role unconfined_r types pong_t;
require
{
   type unconfined_t;
}
domain_type(pong_t)
domain_entry_file(pong_t, pong_exec_t)
domain_auto_trans(unconfined_t,pong_exec_t,pong_t)
```

First our new types for `pong` are defined. This is necessary for being able to use them. `pong_exec_t` is the type that the executable `/bin/pong` is to be labeled with and `pong_t` is the domain that pong should run in. After that we allow the role `unconfined_r` to transition to the domain `pong_t`. The last 3 lines take care of the domain transition. First it is announced that the type `pong_t` is a domain. This is necessary because the policy checker, who is called on compiling the policy, only allows transitions to domain types. Then the following two lines allow the transition and request an automatic transition to the domain `pong_t`.

Besides, the `pong.fc` file is needed to set up the correct security context. It only contains the following line:

```
/bin/pong          --
gen_context(system_u:object_r:pong_exec_t,s0)
```

After compiling this first version of the module and loading it, we could start the actual policy development. With SELinux running in permissive mode we executed pong. The logs of denied accesses in the `pong_t` domain showed us which permissions pong needs and we could use this to write the actual policy. After some evaluating and testing we got a policy that allows pong to operate and at the same time prevents pong from causing harm to the system when its vulnerability is attacked. The policy has the following allow rules. First pong needs to be allowed to be used capabilities in order to be run with `setuid`.

```
# use capabilities
allow pong_t self:capability { setuid net_rawnet_admin};
```

The following three rules allow pong to read configuration files. The first rule allows pong to search the `/etc` directory, the second rule allows to read standard configuration files in `/etc` and the third rule allows to read network related configuration files like `/etc/resolv.conf`

```
# accessing configuration files, e.g. /etc/resolv.conf
allow pong_t etc_t:dir search;
allow pong_t etc_t:file { read getattr };
allow pong_t net_conf_t:file { read getattr };
```

The following rules allow pong to use shared libraries.

```
# use libraries
allow pong_t ld_so_cache_t:file { read getattr };
allow pong_t lib_t:dir search;
allow pong_t lib_t:file { read getattr execute};
allow pong_t lib_t:lnk_file read;
```

Since pong writes output to the terminal, we have to allow this, which is done with the following rules. Besides, we allow pong to be terminated with `Ctrl+c` over the terminal, which is done in the third rule.

```
# accessing the terminal
allow pong_t unconfined_tty_device_t:chr_file { read
write getattr ioctl};
allow pong_t getty_t:fd use;
allow pong_t unconfined_t:process sigchld;
```

The following rules are necessary for pong to work properly. They allow the creation of rawip sockets and sending and receiving of rawip packets.

```
# sending and receiving ICMP packets
allow pong_t self:rawip_socket { write getopt create
read setopt ioctl bind };
allow pong_t self:packet_socket {read write ioctl bind
create connect getattr setopt};
allow pong_t netif_t:netif { rawip_send rawip_recv
udp_send udp_recv};
allow pong_t node_t:rawip_socket node_bind;
allow pong_t node_t:node rawip_recv;
allow pong_t lo_node_t:node rawip_recv;
```

Additionally, we allow creation of UDP sockets and sending and receiving of UDP packets. This is needed if pong needs to use DNS.

```
# DNS
allow pong_t self:udp_socket { read write create connect
getattr setopt};
allow pong_t node_t:node udp_recv;
allow pong_t dns_port_t:udp_socket recv_msg;
```

Finally, we allow pong to receive packets that have not been processed with IPSEC. This must be allowed since pong does not process its ICMP packets via IPSEC.

```
# allow receiving unlabeled packets, refers to IPSEC
allow pong_t unlabeled_t:association recvfrom;
```

With this policy compiled and loaded, we can set SELinux in the enforcing mode again to show that our policy is applicable for preventing the vulnerability in pong from causing harm. But first we demonstrate that pong can really be exploited to get a root shell.

```
danes231@pong3:~/
pong/src$ id
uid=15083(danes231) gid=15083(danes231)
groups=15083(danes231)
context=unconfined_u:unconfined_r:unconfin
ed_t:s0
danes231@pong3:~/
pong/src$ /bin/pong I
`cat nopsled shellcode returns`
224.224.224.224
ping: unknown iface
[illegible]
sh3.2#
id
uid=0(root) gid=15083(danes231)
groups=15083(danes231)
context=unconfined_u:unconfined_r:pong_t:s
0
```

Here we can see that a new shell has been started. By checking the id of the current user, we can confirm that we have got a root shell. Now we put SELinux in the enforcing mode. We need to be root for that.

```
danes231@pong3:~/
pong/src$ su
Password:
pong3:/
home/danes231/pong/src#
setenforce 1
pong3:/
home/danes231/pong/src# exit
```

Now it is time to test, if SELinux can prevent successful attacks on the vulnerable pong command. We run pong with the same arguments that have given us a root shell before.

```
danes231@pong3:~/
pong/src$ /bin/pong I
`cat nopsled shellcode returns`
224.224.224.224
ping: unknown iface
[illegible]
type=1400 audit(1241358474.990:38): avc:
denied { search } for pid=1354
comm="pong" name="bin" dev=ubda ino=18294
scontext=unconfined_u:unconfined_r:pong_t:
s0 tcontext=system_u:object_r:bin_t:s0
tclass=dir
type=1400 audit(1241358474.990:39): avc:
denied { search } for pid=1354
comm="pong" name="bin" dev=ubda ino=18294
scontext=unconfined_u:unconfined_r:pong_t:
s0 tcontext=system_u:object_r:bin_t:s0
tclass=dir
type=1400 audit(1241358474.990:43): avc:
denied { search } for pid=1354
comm="pong" name="bin" dev=ubda ino=18294
scontext=unconfined_u:unconfined_r:pong_t:
s0 tcontext=system_u:object_r:bin_t:s0
tclass=dir
^C
danes231@pong3:~/
pong/src$ id
uid=15083(danes231) gid=15083(danes231)
groups=15083(danes231)
context=unconfined_u:unconfined_r:unconfin
ed_t:s0
```

We see that SELinux already prevents the shellcode from searching the `/bin` direcory for a shell. So the attack is stopped at the first step. If we check the current user, we see that it is still the user we started pong with. Finally we want to make sure that pong still can fulfill its intended tasks.

```
danes231@pong3:~/
pong/src$ /bin/pong
localhost
PING localhost(127.0.0.1) 56(84) bytes of
data.
64 bytes from localhost (127.0.0.1):
icmp_seq=1 ttl=64 time=0.073 ms
64 bytes from localhost (127.0.0.1):
icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from localhost (127.0.0.1):
icmp_seq=3 ttl=64 time=0.059 ms
^C
localhost
statistics 3
packets transmitted, 3 received, 0%
packet loss, time 2019ms
rtt min/avg/max/mdev =
0.057/0.063/0.073/0.007 ms
danes231@pong3:~/
pong/src$ /bin/pong
www.google.de
PING www.l.google.com(209.85.137.104)
56(84) bytes of data.
64 bytes from mginf104.
google.com
(209.85.137.104): icmp_seq=1 ttl=242
time=35.3 ms
64 bytes from mginf104.
google.com
(209.85.137.104): icmp_seq=2 ttl=242
time=35.2 ms
64 bytes from mginf104.
google.com
(209.85.137.104): icmp_seq=3 ttl=242
time=35.3 ms
64 bytes from mginf104.
google.com
(209.85.137.104): icmp_seq=4 ttl=242
time=35.7 ms
64 bytes from mginf104.
google.com
(209.85.137.104): icmp_seq=5 ttl=242
time=36.6 ms
^C
www.
l.google.com statistics 5
packets transmitted, 5 received, 0%
packet loss, time 4063ms
rtt min/avg/max/mdev =
35.226/35.659/36.637/0.571 ms
```

This shows that our policy really does what it should. It grants only those privileges to pong that it really needs and thus prevents the exploitation of the vulnerability. At the same time pong can still execute its desired tasks.

6. Conclusions

In this project we got a picture of SELinux in theory by studying relevant literature and in practice by implementing a security policy for lab systems. Although we only used the basic features of SELinux, we can say that SELinux provides a significant gain in security compared to normal Linux.

Furthermore, the well-known criticism that SELinux be too complex and difficult to configure is in our opinion no longer applicable. The targeted policy, which is by default included in Debian, makes it possible to increase the security of the system bit by bit with the system always keeping usable.

With this in mind, SELinux can be considered to be an effective and feasible way of increasing a system's security.

References

[1] Stephen Smalley, “Configuring the SELinux Policy”, revised 2005 http://www.nsa.gov/research/_files/selinux/papers/policy2.pdf

[2] Red Hat SELinux Guide, 2005 http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide/index.html

[3] Fedora 10, Security enhance Linux User Guide, 2008 http://docs.fedoraproject.org/selinux-user-guide/f10/en-US/index.html

[4] Red Hat Enterprise Linux Deployment Guide, 2008 http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/Deployment_Guide-en-US/index.html

[5] Chris Runge, “SELinux: A New Approach to Secure Systems”, 2004 http://www.redhat.com/f/pdf/sec/WHP001USselinux.pdf

[6] Debian Wiki, SELinux http://wiki.debian.org/SELinux/Setup

[7] Wikipedia, Mandatory Access Control http://en.wikipedia.org/wiki/Mandatory_access_control