Protecting confidential files using SE-Linux

Giuseppe “Gippa” Paternó
Centre for Telecommunications Value-Chain Research
Trinity College Dublin
Dublin 2, Ireland
paternog@cs.tcd.ie

Abstract

A mandatory access control system, such as SE-Linux, is able to protect a computer system containing files and confidential data. Is not an easy task to manage a system protected with SE-Linux as not only it requires strong skills on Unix/Linux, but also high skills on security, mandatory access control systems and how SE-Linux works in deep. Most of the time, however, such level of security is not required and is difficult to find such high-level skills.

This paper, based on a real case, wants to demonstrate that is possible to mediate between a high security environment and a traditional Unix system administration techniques. The challenge was, and still is, to be able to protect highly confidential PDF documents while not teaching SE-Linux to operators and not changing their way of administering the systems.

I. Preface: The ”Defense in Depth” philosophy

Defense is always been a technique, or better an art, since ancient Romans. In the medieval era, precious manuscripts were protected by several layers of defense: they were usually preserved in castles placed in the highest hills, surrounded by a ditch full of crocodiles, in the highest tower, in a secret room, protected by several traps and with poisons inside. Although an attacker may find it easier to breach the first layers, as he advances he continues to meet effective resistance. As he penetrates deeper, the attacker becomes vulnerable and, should the advance stall, the attacker himself risks being entrapped.

Although media, attackers and things to protect have changed, defense techniques still remain valid. In the IT field, it is needed to secure systems and data by using several security layers. We will call this philosophy Defense in Depth[7].

In brief, the Defense in Depth philosophy represents the use of multiple computer security techniques to mitigate the risk of a particular defense being compromised or circumvented. An example could be anti-virus software installed on individual workstations while there is a virus protection on all firewalls and servers within the same environment. Different security products may be on different vectors within the network, helping prevent a shortfall in any one defense leading to a wider failure.

This philosophy should be applied on every system, especially on those that hold sensible data or whenever strict-security is required (military/government/finance/...). Every system, configuration, application and even software development itself has to be thought in a secure way, i.e. giving the least privilege to the user/application, restricting who can do what.

Applying the philosophy in a computing environment means breaking security into the following components:

- Installing a minimised system
- Hardening the system
- Activating/configuring a mandatory access control system (if available) such as SE-Linux
- Use virtualization to confine the application
- Configuring the application securely
- Authenticating users
- Updating systems with the latest security patches

II. The project

Users are always asking if it is possible to leverage SE-Linux to protect given areas of the filesystem or given applications, comparing SE-Linux to a sort of ”application firewall”. Such comparison is conceptually wrong, as SE-Linux is something different.

Security Enhanced Linux, or SE-Linux, is a set of patches to the Linux kernel initially developed by the U.S. National Security Agency (NSA) to bring a Mandatory Access Control (MAC) on Linux, similar to other Trusted Unix platforms such as Trusted Solaris. SE-Linux relies on the modularity of the security architecture of the Linux kernel, referred as Linux Security Modules (LSM): Linux is in fact able to support other security systems such as AppArmor[2]. More about the SE-Linux project and its history can be found on the project website[1].

It is worth to introduce a real case that can be easily adapted to those environments or architectures that must hold confidential files. The objective of the project was to protect highly confidential PDF files: documents had to be delivered to the end-users via a web application with a strong authentication and authorisation mechanism, but system users -including the administrators- were forbidden to access the PDF files in any way. The application is currently placed on a three-node balanced cluster whose files are accessed via GFS[3], the clustered file system that is included in Red Hat Enterprise Linux (see figure 1). In simpler scenarios, a less enforcing method such as DAC[4] (i.e. filesystems permissions and Posix ACLs) might be sufficient, but this project required that a military-grade protection must be ensured and access logs had to be kept.

Note: the protection of PDF files themselves was not taken in in consideration. A good ”Defense in Depth” philosophy, in fact, would protect each individual layer and not only the operating system. Consider that if an attacker is able to exploit a web application application flaw, he/she will be able to download the PDF in clear-text without any protection. A good best practice would be to encrypt the PDF with a random password (and notify the end user in a secure manner) or use a digital certificates infrastructure.

The requirements were:

- Unique identification of users (through LDAP aceess)
- Centralised syslog for audit/security
- Every administrator should login in using exclusively his/her own user id.
- Administrators cannot become root, they should only run the administration commands through sudo [8]
- Administrators should have different access privileges, ranging from a system operator to a full administrator
- Users can not use ”su” command to switch to root even if they know the root password, with the exception of an authorised group
- No user, except root and the application user appserv, can access the directory ”/documents/” and related files/subdirectories:
  - The root user has the right to access the directory, but any read operation must be audited.
  - The application user should not be under audit for performance issues.
  - The application server and some batch processes which need to access the files must run under the appserv

III. The implementation

The challenge of this project was to achieve a compromise between security and manageability of the system. Two types of SE-Linux policies are available in Red Hat derived distributions, such as Fedora, CentOS and Red Hat Enterprise Linux: *targeted* and *strict*. The targeted policy protect and confine only some given daemons, leaving open everything that is not explicitly confined (what is not explicitly confined is named under unconfined_t). Under the strict policy, each program and each environment is confined, having to set the appropriate permissions (called domain transition).

Choosing between the two methodologies is not an easy task: the former is certainly easier to manage, but has problems to ”confine” the system administrators, while the latter offers a military-level security, but needs high-skills to manage the systems, which typically is not common among operators. The choice was a compromise between the two system, by using SE-Linux in targeted mode and leveraging Multi-Category Security (MCS). MCS is the ability to assign a file to a particular category and give each user the right to read/write access to that category, in accordance with the concept of ”no read up, no write down” as described in the Bell-La Padula model[5]. For example, files can be divided in categories such as ”Confidential,” ”Top Secret” or ”HR Department” and the user *foo* is limited to access some of those categories.

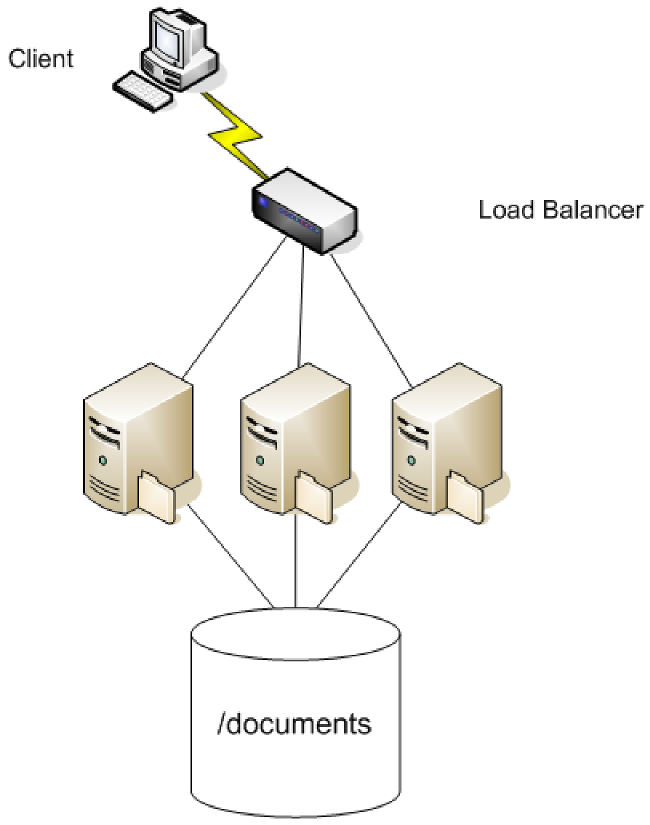

Fig. 1: Architecture overview

This document assumes the following:

- Minimal installation of the operating system
- Hardening of the system, including the authentication mechanisms
- Activation of SE-Linux with the targeted policy: in file /etc/sysconfig/selinux ensure you have the following lines:

```
SELINUX = enforcing
SELINUXTYPE = targeted
```

IV. Definition of the category

An MCS category needs to be defined to label both the directories and the confidential files, in this paper it is named as *TopSecret*. This definition has to be implemented in the file /etc/selinux/targeted/setrans.conf by adding the following line:

```
s0:c3 = TopSecret
```

All files and directories must have this label in order to be protected properly. To give this attribute, use the command *chcat* as the following example:

```
# chcat +TopSecret <filename>
```

The command above is to be executed against each file to be protected: this could be problematic if the application cannot be modified to implement the execution of the command. As such, it is best to leverage the file-context mechanism of the SE-Linux module, so that a simple *restorecon* recursive command will be sufficient, as described in section V.

V. Implementing the SE-Linux module

On Red Hat-derived distributions, the package *selinux-policy-devel* is required in order to have the SE-Linux modules development environment. Not every machine needs to have the development environment, as the package is simply required to compile the policy. Before going further in creating and installing the SE-Linux module, SE-Linux must be temporarily disabled (also referred as *enforcing*) by placing the system in permissive mode with the following command:

```
# setenforce 0
```

The difference between enforcing and permissive mode is that the first ”enforces” the policies by denying what is not permitted, while the second is a way to test the policy and any potential denial is logged but not enforced.

Note: Once finished testing, always remember switch back to the enforcing mode through the command *setenforce 1*. The status of SE-Linux can be verified with the *sestatus* command.

To build the module, create a directory, eg: docsecret, and the following text files within:

- *docsecret.te*, which contains the policy and the definition of new types
- *docsecret.fc*, which contains which files have a given the context

The source of both files are in Appendixes A and B. The file *docsecret.te* contains the definition of the *docsecret_t* type that will help to uniquely identify (or label) the files, protecting them from the confined daemons as defined in the ”targeted” policy of SE-Linux. Furthermore, it contains the permission that the system is required to have in order to attribute the docsecret_t label (in the case of autorelablel of the filesystem or the restoreco command). To compile the module, use the following command:

```
# Make -f /usr/share/selinux/devel/Makefile
```

To load the newly created policy, use the command below:

```
# semodule -i docsecret.pp
```

The module will be permanent across reboots as the *semodule commands* will actually copy the resulting *selinux.pp* file in the */etc/selinux/targeted/modules/active/modules* directory. The command will also reload the new policy sets, therefore loading it into memory. The command *semodule -l* will show the list of the loaded SE-Linux modules. There are two ways to give the correct attributes to the SE-Linux directory/files: use the autorelabel function or the restorecon command. The autorelabel function will be activated during the boot process and its task is to restore the SE-Linux labels according to the file-specified contexts in the active SE-Linux policy. To use the autorelabel function:

```
# touch /.autorelabel
# reboot
```

During the startup process, the init scripts will attribute the labels to files. The scripts are based however on the *restorecon* command, that can be invoked directly. The command takes the settings of the file-context, as they are described on the in-memory SE-Linux policy, and applies them to the file or recursively on a directory, for example:

```
# restorecon -ivvr /documents
```

The meaning of the above switches is: ”-r” means recursively, -v (or more v letters) increase the verbosity of output, ”-i” to ignore files that do not exist.

If a user attempts to see the content of the root directory with a non-root user, such as the admin user, with the command ”ls -laZ /”, he/she will notice that next to the words ”documents” some question marks are displayed. This is the proof that users have no access to the files labeled TopSecret. If a user tries to go to the document directory, with ”cd /documents”, even if the permission of the directory is 644, the system will deny the command with a ”permission denied”.

VI. Enabling the application

The above configuration confined the user, the next step is to enable the user appserv, with which we will run the application server, to access the protected files.

By default, all users have no access to any file that has a category associated with the exception of the root user. To enable a user to access the label TopSecret, such as user appserv, execute the following command:

```
# chcat -l +TopSecret appserv
```

The command to verify that the user has been assigned the correct level is *semanage* as used in the following example:

```
# semanage login -l
Login Name          SELinux User MLS/MCS Range
__default__         user_u       s0
appserv             user_u       -TopSecret
root                root         SystemLow-SystemHigh
```

The chcat command acts as a wrapper for the commands chcon and semanage; please check the man pages for more on the commands.

VII. Customising sudo

Administrators must be able to perform their day-by-day tasks by running the programs that are appropriate for their role, without being root. This is accomplished by making use of the sudo command. In the described scenario, a common operator should be able start/stop the Jboss container, as well as issuing reboot and poweroff commands. It should be noted that running those commands as root does not mean that the SE-Linux labels are ineffective, therefore the operator wont be able to access and/or perform operations on the confidential documents, whether the command is executed as root or not. A simple demonstration can be accomplished by adding in /etc/sudoers the following:

```
admin ALL = NOPASSWD: /usr/bin/id
admin ALL = NOPASSWD: /bin/cat
```

If an user logs on the system, for example using the admin user, and type the command ”sudo id” he/she will get the following result:

```
$ sudo /usr/bin/id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),
6(disk),10(wheel) context=user_u:system_r:unconfined_t
```

Note the context associated with the admin user. By default the root user has the context root:system_r:unconfined_t:SystemLow-SystemHigh. The admin user, even after becoming root with sudo (uid = 0), maintained the SE-Linux user context (user_u). An interesting test that can be performed is the following:

```
$ sudo cat /documents/test
cat: /documents/test: Permission denied
```

Note that, despite becoming root, the user can't access to SE-Linux protected directory.

The goal has been accomplished, i.e. being able to manage the system in a simple and comfortable way, while maintaining the confidentiality of the files hold in the directory /documents/. Any denied access is logged with a message in the audit subsystem, i.e. the file /var/log/audit/audit.log.

Note: Users must not be allowed to execute any command via sudo! Administrators must not be able to access the SE-Linux subsystem or the audit system, in particular the setenforce and auditctl commands. Permit the single command, or group of commands, that are meant to administer the system instead.

VIII. Further "root" restrictions

The root user is still active and has the ability to execute strategic commands, such as put SE-Linux in permissive mode or even disable the audit subsystem. So it is a best practice to avoid direct root user access:

- Changing the behaviour of the "su" command is necessary
- Also the ssh daemon must be configured accordingly

As per initial requirements, a group of people can still use the "su" command. Deleting or disabling completely the "su" command can lead to problems as some cron jobs and batch scripts rely on the command itself. The *wheel* group can be used to restrict the command as it's already in the system and there's an ad-hoc module for it (pam_wheel). Set up the associated pam file for the su command, i.e. /etc/pam.d/su, as follows:

```
#% PAM-1.0
auth       sufficient pam_rootok.so
auth       required   pam_wheel.so use_uid
auth       include    system-auth
account    sufficient pam_succeed_if.so uid=0 use_uid quiet
account    include    system-auth
password   include    system-auth
session    include    system-auth
session    optional   pam_xauth.so
```

With this configuration, only those users belonging to the wheel group are allowed to run the command "su", even if other users know the root password. The next step is to disable direct access to root via SSH, acting on /etc/ssh/sshd_config and setting the "PermitRootLogin no" parameter. Also, a best practice would restrict the users that are allowed to log-in interactively via ssh by using the options AllowUsers and/or AllowGroups.

The root account can further be restricted by specifying in /etc/securetty the console and the virtual console on which root can connect.

IX. Audit Subsystem

By default, SE-Linux keeps track of all attempts to access unauthorized resources (deny logs), such as if the admin user tries to access the "/documents" directory. Another requirement is to log root attempts to access the documents directory: a root user that attempts to access a number of PDF files from a console can be suspicious.

The Linux operating system has a very flexible audit subsystem that is able to intercept system calls and log them into a file. Through an audit module, named audispd[6], it is possible to redirect the log of the audit subsystem to syslog, so that it is possible to send the data to a remote syslog. Imagine the case in which a malicious root user can log in locally, copy some PDF files and delete the log files to cover his/her steps: in such configuration, any log is sent to a central logging server will record the attempts.

Such configuration is done in the /etc/audit/audit.rules file by inserting the following line:

```
-a exit,always -S open -S truncate -F dir=/documents -F uid != 300
```

In brief, this command keeps track of the open and truncate system calls (syscall) to the /documents directory, only if the userid is not equivalent to UID 300. UID 300 is the UID that is associated to the appserv user. With this configuration, any user attempt to access the directory containing the confidential files will be logged. Even if administrators/operators cannot access the files because they are "armored" by the SE-Linux policy, this rule allows administrators to log each access by authorized users (eg: root). To send these logs to syslog (audispd), set "active = yes" in the file /etc/audisp/plugins.d/syslog.conf.

X. Using ACL

An authorised root account user can accidentally disable SE-Linux from console, for example for troubleshooting reasons. When the system is disarmed, any access to the confidential documents can be performed unless proper permissions have been set. For this reason, normal DAC should also be used on top of SE-Linux, by applying an ACL to the /documents directory with the following commands:

```
# chmod 0750 appserv:appserv /documents/
# setfacl -m appserv:rwx /documents/
# setfacl -m root:rwx /documents/
# getfacl --access /documents/ | setfacl -d -M- /documents/
```

XI. Conclusions

SE-Linux is a very interesting and powerful subsystem, yet it is very difficult to manage even for experienced users. This paper demonstrates that is possible to achieve a compromise between security and system manageability, so that administrators can introduce a mandatory access control system in their environment with a higher degree confidence.

The key is to use the right technologies in the right context: using SE-Linux has fewer practical advantages in a database system whose access is only done via the database protocol and interactive ssh access is restricted only to administrators. However, it can be a good idea to use SE-Linux armored systems in front-end internet systems or where there's a need for confidential files to be protected, as the case described in this paper.

The author wishes to stress how important is to adopt the "Defense in Depth" philosophy: security should become stricter as it approaches to data that has to be protected.

XII. Acknowledgements

Many thanks to Prof. Donal O'Mahony, director of CTVR at Trinity College Dublin (Ireland), that helped me in publishing this paper. Many thanks to my customers, especially Anna Petrucci and Attilio Sergio Matteucci, that trusted me on this project. Also thanks to my wife Maria for her patience and Ezio de Mauro for helping me on the translation.

XIII. Appendix A: docsecret.te

```
policy_module(docsecret,1.0.20)

require {
        type unconfined_t;
        type restorecon_t;
        type setroubleshootd_t;
        type fs_t;
        type default_t;
}

type docsecret_t;
files_type(docsecret_t)

## for allowing restorecon
allow docsecret_t fs_t:filesystem associate;

allow restorecon_t docsecret_t:dir { read relabelfrom relabelto getattr };
allow restorecon_t docsecret_t:file { read relabelfrom relabelto getattr };
allow unconfined_t docsecret_t:dir rw_dir_perms;
allow unconfined_t docsecret_t:file rw_file_perms;
```

XIV. Appendix B: docsecret.fc

```
##
## Main document directory
##

/documents(/.*)?                  gen_context(system_u:object_r:docsecret_t,s0,c3)
```

References

[1] SE-Linux is hosted on the NSA site at http://www.nsa.gov/selinux/

[2] AppArmor is available at http://forge.novell.com/modules/xfmod/project/?apparmor

[3] GFS supports extended attributes (xattr), which usually allow the storage of information on the ACL. SE-Linux uses the extended attributes to write the information on file-contexts. If you want to use another filesystem, you need to know whether this supports the extended attributes.

[4] Discretionary Access Control (DAC) that is normally present in the Unix operating system, as opposed to the Mandatory Access Control

[5] More info on the Bell-Lapadula model at http://en.wikipedia.org/wiki/Bell-LaPadula_model

[6] The plugin audispd for syslog is available in the package audispd-plugins

[7] Defense in Depth, A practical strategy for achieving Information Assurance in todays highly networked environments. On-line at http://www.nsa.gov/ia/_files/support/defenseindepth.pdf

[8] Sudo (su ”do”) allows a system administrator to delegate authority to give certain users (or groups of users) the ability to run some (or all) commands as root or another user while providing an audit trail of the commands and their arguments. More at http://www.sudo.ws/