SELinux

DAC vs MAC:

Security-Enhanced Linux (SELinux, https://github.com/SELinuxProject) is a kernel module that implements a Mandatory Access Control (MAC) system. It was developed as a replacement for Discretionary Access Control (DAC) that ships with most Linux distributions. If you are only using a DAC system (for instance, the standard permissions/ACLs model) :

*Administrators have no way to control users: a user could set world readable permissions on sensitive files (such as ssh keys). They could even *chmod +rwx* their entire home directory and nothing would stop them. The same reason can be extended to any (maybe trojaned) program executed by that user because processes inherit user's right. In summary: a regular user can grant (and restrict) access to their owned files to other users and groups or even change the owner of the file, leaving critical files exposed to accounts who don't need this access. It's true that this regular user could also restrict to be more secure, but that's discretionary: there's no way for the system administrator to enforce it for every single file in the system.

*Root (or sudo) access on a DAC system gives the (maybe hacked) person or the (maybe trojaned) program permission to perform as desired on a machine. In essence, there are two privilege levels (root and user), and no easy way to enforce a model of least-privilege (chroot jails are not an option because they are discretionary too).

But if you are using a MAC system, users won't be able to work around the rules previously set by the system administrator. These rules define what a user or process can do, confining every process to its own domain so the process can interact with only certain types of files and other processes from allowed domains. This prevents a hacker from hijacking any process to gain system-wide access. This restriction is implemented from the kernel level: it's enforced as the SELinux policy loads into memory, and thus the access control becomes mandatory.

In SELinux everything is denied: a series of exceptions policies must be written by sysadmin to give each element of the system (a service, process type or user) only the access required to function (to specific files, ports, pipes, ...). If a service, program or user subsequently tries to access or modify a file or resource not necessary for it to function, then this is is denied and the action is logged. For instace, the Apache user can access only the /var/www/html directory and nothing more because there's a SELinux policy which mandates that. There is no access by default, even if you're the root user.

Because SELinux is implemented within the kernel, individual applications do not need to be especially written or modified to work under SELinux although, of course, if written to watch for the error codes which SELinux returns might work better afterwards. If SELinux blocks an action, this is reported to the underlying application as a normal (or, at least, conventional) "access denied" type error to the application.

SELinux policy is not something that replaces traditional DAC security. If a DAC rule prohibits a user access to a file, SELinux policy rules won't be evaluated because the first line of defense has already blocked access. SELinux security decisions come into play after DAC security has been evaluated. So if access to a resource is denied, please check access rights first. But note that if the DAC and MAC conflict, the SELinux policy takes priority. So, let's say that you change —as root— the ownership of httpd service to anyone (by running the command chmod 777 httpd); the default SELinux policy still prevents any random user from killing the web server.

A great introduction to SELinux is https://wiki.gentoo.org/wiki/SELinux/Quick_introduction and, in general, whole https://wiki.gentoo.org/wiki/SELinux site.

Activació de SELinux:

SELinux has two operating modes: permissive and enforcing. Permissive mode allows the system to function by using its regular DACs systems while logging every violation to SELinux. The enforcing mode enforces a strict denial of access to anything that isn't explicitly allowed by SELinux policies. You can see which mode system is running by executing this command:

getenforce

Another command which can be used to know the same (though it shows other informational data) is

sestatus

NOTA: The meaning of the information shown by this command, among other interesting arguments, is explained in https://www.thegeekstuff.com/2017/06/selinux-sestatus/

You can inmediately change to the desired mode simply executing...

sudo setenforce permissive (or *sudo setenforce enforcing*)

If you want this change to be permanent, you should modify the "**/etc/selinux/config**" file (which is linked by "/etc/sysconfig/selinux"). Specifically you should have this line...

SELINUX=permissive (or *SELINUX=enforcing*)

...and then rebooting the system.

By design, SELinux allows different policies to be written that are interchangeable by editing the SELINUXTYPE= line in "/etc/sysconfig/selinux" file. The default policy is the "targeted" policy which confines selected previously targeted system processes. By default many system processes are already targeted in RedHat family systems from its installation (including httpd, named, dhcpd, mysqld, etc) but all other system processes and all remaining userspace programs runs in an unconfined domain and is not covered by the SELinux protection model. One goal might be to target every process to force them to be run in a confined domain so the "targeted" policy protects as many key processes as possible, but it's responsability from package maintainers. The alternative would be a deny-by-default model where every access is denied unless approved by the policy: it would be a very secure implementation, but this also means that developers have to anticipate every single possible permission every single process may need on every single possible object. So, in summary ,the default behaviour sees SELinux concerned with only certain processes.

The targeted SELinux policy ships with 4 forms of access control:

*Type Enforcement (TE): Type Enforcement is the primary mechanism of access control used in the targeted policy
*Role-Based Access Control (RBAC): Based around SELinux users (not necessarily the same as the Linux user), but not used in the default configuration of the targeted policy
*Multi-Level Security (MLS): Not commonly used and often hidden in the default targeted policy.
*Multi-Category Security (MCS): An extension of Multi-Level Security, used in the targeted policy to implement compartmentalization of virtual machines and containers through sVirt.

Conceptes bàsics de SELinux:

An SELinux policy defines user access to roles, role access to domains, and domain access to types. First the user has to be authorized to enter a role, and then the role has to be authorized to access the domain. The domain in turn is restricted to access only certain types of files:

*SELinux has a set of pre-built users. Every regular Linux user account is mapped to one or more SELinux users. We will study this.

*A role is only used when SELinux is configured to implement RBAC (so it's not used by default). A role is a filter: it defines what users can assume the role in question and what domain this role can access to.

*A domain is the context within which a SELinux process can run. That context is like a wrapper around the process: it tells the process what it can and can't do. For example, the domain will define what files, directories, links, devices, or ports are accessible to the process.

*A type is the context for a file that stipulates the file's purpose. For example, the context of a file may dictate that it's a web page, or that the file belongs to the "/etc" directory, or that the file's owner is a specific SELinux user.

The policy itself is a bunch of rules that say that so-and-so users can assume only so-and-so roles, and those roles will be authorized to access only so-and-so domains. The domains in turn can access only so-and-so file types.

Terminology tip 1: In general, when we call something as a SELinux "subject" we will referring to a process that can potentially affect an object. An SELinux "object" is anything that can be acted upon: this can be a file, a directory, a port, a tcp socket, the cursor, or perhaps an X server. The actions that a subject can perform on an object are the subject's permissions.

Terminology tip 2: The last step, where a process running within a particular domain can perform only certain operations on certain types of objects, is called Type Enforcement (TE).

Paràmetre *-Z* i comandes bàsiques SELinux: *semanage, chcon, restorecon* :

As we have just said, the SELinux primary model or enforcement is called "type enforcement". Basically this means we define a "label" on a process based on its type, and a "label" on a file system object based on its type, too. Policy rules then control access between labeled subjects (processes, daemons) and labeled objects (folders, files, devices, ports...) -or more specifically, they controls which users can get which roles and each specific role places a constraint on what type of files that user can enter-.

These labels (some times called "contexts" too) are stored as extended attributes on the file system or, for objects that aren't files (such as processes and ports) are managed by the kernel. The label format is "user:role:type:sensitivity". Let's see the label/context of a file (for instance, the one of the "/etc/fstab" file) using the -Z parameter of ls command:

ls -lZ /etc/fstab

In addition to the standard file permissions and ownership, we can see the SELinux security context fields: "system_u:object_r:etc_t:s0". As you can see, SELinux Users are suffixed by "_u", roles are suffixed by "_r" and types (for files) or domains (for processes) are suffixed by "_t".

NOTA: The -Z switch will work with most utilities to show SELinux security contexts (*ps -eZ, id -Z, cp -Z, mkdir -Z, ss -Z*)

*Each Linux user account maps to an SELinux user, and in this case, the root user that owns the file is mapped to the system_u SELinux user. This mapping is done by the SELinux policy.

*The second part specifies the SELinux role, which is object_r. We talk about roles later

*What's most important here is the third part, the type of the file that's listed here as etc_t. This is the part that defines what type the file or directory belongs to. We can see that most files belong to the etc_t type in the /etc directory. We can also see some files may belong to other types, like locale.conf which has a locale_t type. Even when all the files listed here have the same user and group owners, their types could be different.

*The fourth part of the security context, s0, has to do with multilevel security or MLS. Basically this is another way of enforcing SELinux security policy different from "targeted" one, and this part shows the sensitivity of the resource (s0). We won't talk about this because it's a very advanced topic.

If we look at the SELinux security context of a file in our home directory we will see something like this: "unconfined_u:object_r:user_home_t:s0" (so we can infer that the type user_home_t is the default type for files in a user's home directory).

We can see the label of a (not running) binary, too: if we do *ls -lZ /bin/ssh* we will see its context is "system_u:object_r:ssh_exec_t:s0"

The user, role, and sensitivity work just like the same contexts for files than for processes. Instead, the domain is unique to processes. A domain gives the process a context to run within: it's like a bubble around the process that confines it. This confinement makes sure each process domain can act on only certain types of files and nothing more. Using this model, even if a process is hijacked by another malicious process or user, the worst it can do is to damage the files it has access to. If you want to know the SELinux security context of a process; for instance, the journald daemon (systemd-journald), you could do...:

ps -eZ | grep "systemd-journald"

...and you will see the SELinux context fields now are: "system_u:system_r:syslogd_t:s0" , so its domains is "syslogd_t".

Access is only allowed between similar domains and types: a process running in the httpd_t context, for instance, can interact with an object with the httpd_*something*_t label. So, as Apache runs in the "httpd_t" domain it can read "/var/www/html/index.html" because it is of type "httpd_sys_content_t" but it can not access "/home/username/myfile.txt" even though this file is world readable because "/home/username/myfile.txt" SELinux security context is not of type "httpd_t". If Apache were to be exploited, assuming for the sake of this example that the root account right needed to effect a SELinux re-labeling into another context were not obtained, it would not be able to start any process not in the httpd_t domain (which prevents escalation of privileges) or access any file not in an httpd_t related domain.

Unless specified by the policy, processes and files are created with the contexts of their parents. So if we have a process called "proc_a" spawning another process called "proc_b", the spawned process will run in the same domain as "proc_a" unless specified otherwise by the SELinux policy. Similarly, if we have a directory with a type of "some_context_t", any file or directory created under it will have the same type context unless the policy says otherwise. This inheritance is not preserved when files are copied to another location. In a copy operation, the copied file or directory will assume the type context of the target location This change of context can be overridden by the *--preserver=context* clause in the cp command. (available from "setools-console" package)

We can use 'semanage' to change the context of a file/directory (in a similar way to how 'chown' or 'chmod' may be used to change the ownership or standard file permissions of a file). For instance, to change the context of /home/dan/html, run the following commands:

sudo semanage fcontext -a -t httpd_sys_content_t '/home/dan/html(/.)?'*

NOTA: If you don't know the label but you know a file with the equivalent labeling you want, you can do: *sudo semanage fcontext -a -e /path/file1 /path/file2*

To check security context of all files/directories in the filesystem we can execute:

sudo semanage fcontext -l

However, another command that can be used to do the same (that's is: to change a context of a file or files/directories (optionally recursively with *-R* parameter) is 'chcon':

sudo chcon -vR -t httpd_sys_content_t /home/dan/html

NOTA: If you don't know the label but you know a file with the equivalent labeling you want, you can do: *sudo chcon --reference /path/file1 /path/file2*
NOTA: You can change all the context simply doing *chcon user:role:type:sensitivity /path/file* or any other of the parts of the context with *-u* , *-r* or *-l* arguments, respectively.

Modifying security contexts in both manners (via 'semanage fcontext -a' or via 'chcon') will persist between system reboots but in case of having executed the change via 'chcon', it will last only until that modified portion of the filesystem is relabeled. If you want a context change will be permanent, even through a complete filesystem is relabeled, you should use 'semanage fcontext -a'. That is because the latter writes a local custom rule (a so-called Policy Module) to the /etc/selinux/targeted/contexts/files/file_contexts.local file so it can "remembered" later, if it's necessary re-apply it.

The 'restorecon' command may be used to restore file(s) default SELinux security contexts (as they are stored in /etc/selinux/targeted/contexts/files/file_contexts and file_contextx.local file, as we will see soon). Let's use Apache as an example: suppose a user edits a copy of index.html in his/her home directory and moves (mv) the file to the DocumentRoot /var/www/html (which has a different context by default: ‘httpd_sys_content_t’). Whilst the copy (cp) command will typically adopt the default destination directory's or file's security context, move (mv) will maintain the source's security context. So Apache will fail to load this file because it won't have the correct label. We could use the 'chcon' command to change the security context of the file(s) in question but as the file(s) are now in the default Apache DocumentRoot (/var/www/html) we can just restore the default security contexts for that directory or file(s). In fact, running chcon requires you to know the correct context for the file but restorecon doesn’t need this specified. To restore just the index.html file, we would use...:

sudo restorecon -v /var/www/html/index.html

...or to recursively restore the default security contexts for the whole directory:

sudo restorecon -Rv /var/www/html

As we said, the system knows which context to apply when it runs restorecon because SELinux “remembers” the context of every file or directory: contexts of files already existing in the system are listed in the /etc/selinux/targeted/contexts/files/file_contexts file. It’s a large file and it lists every file type associated with every application supported by the Linux distribution. Contexts of new directories and files are recorded in the /etc/selinux/targeted/contexts/files/file_contexts.local file. So when we run the restorecon command, SELinux will look up the correct context from one of these two files and apply it to the target, applying inheritance when necessary.

NOTA: If we simply wanted to examine the security contexts of the /var/www/html directory to see if any files needed their security contexts restored, we can use restorecon with the -n switch to prevent any relabelling occurring: *sudo restorecon -Rv -n /var/www/html* . Other interesting arguments of restorecon are *-f listfilestochange.txt*, *-o listchangedfiles.txt* or *-e /subfolder/to/exclude*

NOTA:There is a nifty tool called *matchpathcon* that can help troubleshoot context-related problems. This command will look at the current context of a resource and compare it with what’s listed under the SELinux context database. If different, it will suggest the change required. Let’s test this with the /www/html/index.html file. We will use the -V flag that verifies the context: *matchpathcon -V /www/html/index.html*

NOTA: Another interesting tool is *seinfo -t* (available from "setools-console" package) which lists all contexts currently in use on your system. grep for the name of your application.

If you want to allow some labeled processes (say, the Apache web server) to run in such a way that it generates SELinux error messages rather than just failing, instead of using 'setenforce permissive' (which puts all domains on the system into permissive mode) you can use following command instead, which puts just the domain specified):

sudo semange permissive -a httpd_t

NOTA: To know what domain you should put into permissive mode (that is, the domain which has denied problems), you could look at the type in the context of the "scontext" field from audit log entries (that is, the ones with the field "type" with the "AVC" and "denied" values associated; more about this later)

Simultaneously, your other processes run under SELinux's restricted mode protection. Once you have that process running without any error messages, you can return it to enforcing mode with:

sudo semanage permissive -d httpd_t

To check on the current SELinux mode status of your processes, run:

sudo semanage permissive -l

NOTA: Per saber més sobre RBAC https://wiki.centos.org/HowTos/SELinux#head-91a597b2b6f140484d62d59a0b9a1dfea4dffc50
NOTA: Per saber més sobre MCS (https://wiki.centos.org/HowTos/SELinux#head-563ca75234163cdfa0ef056f1f82d4d396526d2b
NOTA: Per saber més en general: https://www.linuxtopia.org/online_books/getting_started_with_SELinux/SELinux_overview.html

Booleans SELinux

Booleans allows us to make changes to part of policy at runtime without need for having knowledge of policy writing. This allows changes to be implemented without the need to recompiling a SELinux Policy or adding your own custom SELinux modules with *audit2allow*. For example, let’s say we want to share our user’s home directory over FTP for read-write access & we have already shared them but while trying to access them, we can’t see them. That’s because SElinux policy is preventing the FTP daemon from reading & writing in user’s home directory. We need change the policy so that ftp can access home directories, to do that we will see if there are any Booleans available to accomplish it by running,

sudo semanage boolean –l

It will produce a list of all available Booleans with their current status (on or off) and a description.
NOTE: You can also get list of available Booleans by running *getsebool -a* but it will not show its description

To turn on/off a Boolean you can use the ‘setsebool’ command. For instance, if you execute this...:

sudo setsebool ftp_home_dir on

...now our ftp daemon will be able to access user’s home directory.

Another example: the default behavior is that a web application running in the httpd_t context will not be allowed to send emails. That helps greatly to prevent a vulnerable web application to send out SPAM. Well, if you want to operate a web mail service Apache must be able to send emails. No big deal: *setsebool -P httpd_can_sendmail on*

NOTA: If the *-P* option is given, all pending values are written to the policy file on disk so they'll be persistent across reboots.

NOTA: Hi ha més trucs aquí: https://wiki.centos.org/TipsAndTricks/SelinuxBooleans . On the other hand, a full example interesting to read is https://www.rootusers.com/deploy-a-basic-cgi-application-with-apache

NOTA: Even when a system is in enforcing mode a malicious user with root access can manipulate policies or put SELinux into permissive mode. There is a method to prevent this: lock down your system doing *setsebool -P secure_mode_policyload on* Be aware: Once active nothing can not be changed during runtime, you need to reboot your system and provide selinux=1 enforcing=0 as grub boot parameter to be able to change any SELinux settings.

Troubleshooting SELinux:

Sooner or later you may run into situations where SELinux denies access to something and you need to troubleshoot the issue. There are a number of fundamental reasons why SELinux may deny access to a file, process or resource:

A mislabeled file.
A process running under the wrong SELinux security context.
A wrong policy tuning due to working with booleans.
A bug in policy (an application requires access to a file that wasn't anticipated when the policy was written and generates an error)
An intrusion attempt.

The first 4 we can deal with, whereas giving alarm and notice in the 5th case is exactly the intended behaviour. To troubleshoot any issue, the log files are key and SELinux is no different. By default SELinux log messages are written to /var/log/audit/audit.log via the Linux Auditing System auditd, which is started by default. If the auditd daemon is not running, then messages are written to /var/log/messages. SELinux log messages are labeled with the "AVC" keyword so that they might be easily filtered from other messages, as with grep.

On the other hand, we can install the "setroubleshoot-server" package (and its GUI too, the "setroubleshoot" package) which gives more information to debug the eventual AVC denials. The purpose behind these packages is to let users know when access has been denied, help them resolve it if necessary, and to reduce overall frustration while working through tight security restrictions in the default SELinux policies. Former package provides a service called setroubleshootd (activated by D-Bus, so "dbus" service should be running on system to that setrobuleshootd works fine) and a command called sealert which acts as a "frontend" to this server.

NOTA: You can even notify yourself via email when some AVC happens!. To do so, open "/etc/setroubleshoot/setroubleshoot.conf" file and adjust the [email] section to fit your server:

recipients_filepath = /var/lib/setroubleshoot/email_alert_recipients
smtp_port = 25
smtp_host = localhost
from_address = selinux@myserver.com
subject = [MyServer] SELinux AVC Alert

and then, execute this: echo "destiny@mycompany.com" >> /var/lib/setroubleshoot/email_alert_recipients

A really easy (and safe) test is to ask sshd to bind to a non-standard port. Simply define an additional port on in your /etc/ssh/sshd_config. When you restart sshd, it won't be allowed to bind to port 222 (since that's blocked by SELinux as a non-standard port for the ssh_port_t port type). To see all available information about all SELinux log messages saved in /var/log/audit/audit.log, we can execute:

sudo sealert -a /var/log/audit/audit.log

or the 'ausearch' command instead (which belongs not to SELinux framework but to the audit system). Anyway, you can search then the message_ID with the AVC message number you are interested in and execute this:

sudo sealert -l [message_ID]

You'll see a friendlier explanation of what went wrong, and a suggestion for a fix. In this case, the suggestion would be execute:

sudo semanage port -a -t ssh_port_t -p tcp 222

A full list of ports that services are permitted access by SELinux can be obtained executing:

semanage port -l.

NOTA: Additionally we can use the *-C* flag, which only displais customizations (the full contents of -l are 400 lines!)

If we were instead meant to apply TCP port 2222 to Apache we can use 'semanage' with the *-m* flag to modify the SELinux port type from ssh_port_t to http_port_t as shown below:

sudo semanage port -m -t http_port_t -p tcp 2222

We can also completely delete a record by using the *-d* flag, afterwards the output of *-C* confirms there are no longer any customizations in place:

sudo semanage port -d -t http_port_t -p tcp 2222

NOTA: The *sepolicy* command, which comes with the policycoreutils-devel package, can also be used to view SELinux port types that have been set on a particular port.

<u>Audit commands cheatsheet</u>

Some of the most common searches to find AVC error messages are:

*All error messages: ***ausearch -m avc***
*Today's error messages: ***ausearch -m avc -ts today***
*Error messages from the past 10 minutes: ***ausearch -m avc -ts recent***
*Or, to find SELinux denials for a particular service, use the -c comm-name option, where comm-name is the executable's name. For example, httpd for the Apache web server or smbd for Samba: ***ausearch -m avc -c httpd***

A AVC error message can these fields (among others):

type=AVC and avc: SELinux caches access control decisions for resource and processes. This cache is known as the Access Vector Cache (AVC). These two fields of information are saying the entry is coming from an AVC log and it's an AVC event.
denied { *getattr* **}**: The permission that was attempted and the result it got. In this case the "get attribute" operation was denied.
pid: The process id of the process that attempted the access.
comm: The process id by itself doesn't mean much. The comm attribute shows the process command (httpd, for instance)
path: The location of the resource that was accessed (for instance, a file under /www/html/index.html)
dev and **ino**: The device where the target resource resides and its inode address.
scontext: The security context of the source process's domain (for instance, httpd_t domain).
tcontext: The security context of the target resource (for instance, default_t)
tclass: The class of the target resource (for instance, a file)

Per saber més sobre els possibles camps disponibles, consultar: https://wiki.centos.org/HowTos/SELinux#head-c84fad68bffc5eca190fa3a1aab3cac2cfe94a63

<u>SELinux Policies</u>:

The SELinux security policy for Fedora is shipped in the "selinux-policy" package. The policy rules come in the form: "allow domains types:classes permissions;", where:

*Domain : A label for the process or set of processes.
*Type : A label for the object (e.g. file, socket) or set of objects.
*Class : The kind of object (e.g. file, socket) being accessed.
*Permission : The operation (e.g. read, write) being performed.

We can use the 'sesearch' command (available from "setools-console" package) to check, for instance, the type of access allowed for the httpd daemon to files:

sesearch --allow -s httpd_t -t httpd_sys_content_t -c file

Notice the first line: "*allow httpd_t httpd_sys_content_t : file { ioctl read getattr lock open };*" It says that the httpd daemon has I/O control, read, get attribute, lock, and open access to files of the httpd_sys_content type. We could "filter" even more adding the *-p* argument to specify a permission (it can specified multiple times). From an AVC record in the audit log we could use sesearch to identify the corresponding rule that deny the requested access to the target.

Another real example of such a rule could be: "*allow httpd_t httpd_log_t: file { append create getattr ioctl lock open read setattr };"* The rule allows any process labeled as httpd_t to create, append, read and lock files labeled as httpd_log_t. Remember using the ps command, you can list all processes with their labels (*ps -efZ | grep httpd*) and using 'semanage' you can know which objects are labeled as httpd_log_t (*semanage fcontext -l | grep httpd_log_t*).

'Sesearch' can be useful too to identify any rule that is toggled by booleans: the identifier shown in square brackets is the name of the boolean that would allow this access and the final boolean shown indicates if it is currently on. We might also want to inspect exactly what any boolean is allowing (that is, which rules are equivalent to activate any boolean), and the same sesearch utility lets us do that:

sesearch --allow -b boolean_name [-c class] [-p permission]

SELinux users and roles:

Roles are not that important when we see them in file security contexts. For files, it's listed with a generic value of object_r. Roles become important when dealing with users and processes. SELinux users are different entities from normal Linux user accounts, including the root account. An SELinux user is not something you create with a special command, nor does it have its own login access to the server. Instead, SELinux users are defined in the policy that's loaded into memory at boot time, and there are only a few of these users. The user names end with _u, just like types or domain names end with _t and roles end with _r. Different SELinux users have different rights in the system and that's what makes them useful.

The SELinux user listed in the first part of a file's security context is the user that owns that file. This is just like you would see a file's owner from a regular ls -l command output. A user label in a process context shows the SELinux user's privilege the process is running with.

When SELinux is enforced, each regular Linux user account is mapped to an SELinux user account. There can be multiple user accounts mapped to the same SELinux user. This mapping enables a regular account to inherit the permission of its SELinux counterpart. To view this mapping, we can run following command:

sudo semanage login -l

The first column in this table, "Login Name", represents the local Linux user accounts. Any regular Linux user account is first mapped to the "default" login. This is then mapped to the SELinux user called unconfined_u (this is the second column). The third column shows the multilevel security / Multi Category Security (MLS / MCS) class for the user but for now, let's ignore that part and also the column after that (Service). Next, we have the root user. Note that it's not mapped to the "default" login, rather it has been given its own entry. Once again, root is also mapped to the unconfined_u SELinux user.

'system_u' is a different class of user, meant for running processes or daemons. To see what SELinux users are available in the system, we can run following command:

sudo semanage user -l

What does this bigger table mean? First of all, it shows the different SELinux users defined. We had seen users like unconfined_u and system_u before, but we are now seeing other types of users like guest_u, staff_u, sysadm_u, user_u and so on (the names are somewhat indicative of the rights associated with them). Let’s look at the fifth column, SELinux Roles; if you remember from the first part of this tutorial, SELinux roles are like gateways between a user and a process. We also compared them to filters: a user may enter a role, provided the role grants it. If a role is authorized to access a process domain, the users associated with that role will be able to enter that process domain.

Now from this table we can see the unconfined_u user is mapped to the system_r and unconfined_r roles. Although not evident here, SELinux policy actually allows these roles to run processes in the unconfined_t domain. Similarly, user sysadm_u is authorized for the sysadm_r role, but guest_u is mapped to guest_r role. Each of these roles will have different domains authorized for them.

Now if we take a step back, we also saw from the first code snippet that the default login maps to the unconfined_u user, just like the root user maps to the unconfined_u user. Since the **default_** login represents any regular Linux user account, those accounts will be authorized for system_r and unconfined_r roles as well. So what this really means is that any Linux user that maps to the unconfined_u user will have the privileges to run any app that runs within the unconfined_t domain (as you can imagine, unconfined processes would have all types of access in the system but even then, this full access is not arbitrary: full access is also specified in the SELinux policy). To demonstrate this, let’s run the id -Z command as the root user:

id -Z

This shows the SELinux security context for root: "unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023". So the root account is mapped to the unconfined_u SELinux user, and unconfined_u is authorized for the unconfined_r role, which in turn is authorized to run processes in the unconfined_t domain. If you execute the same id -Z command for a regular user, you will get the same output: a regular user maps to the same SELinux user/role/domain than the root account also maps to.

We had seen the list of a number of SELinux users before:

*guest_u: This user doesn’t have access to X-Window system (GUI) or networking and can’t execute su / sudo command.
*xguest_u: This user has access to GUI tools and networking is available via Firefox browser.
*user_u: This user has more access than the guest accounts (GUI and networking), but can’t switch users by running su or sudo.
*staff_u: Same rights as user_u, except it can execute sudo command to have root privileges.
*system_u: This user is meant for running system services and not to be mapped to regular user accounts.

To see how SELinux can enforce security for user accounts, let’s think about a "regularuser" account. As a system administrator, you now know the user has the same unrestricted SELinux privileges as the root account and you would like to change that. Specifically, you don’t want the user to be able to switch to other accounts, including the root account. So we will change regularuser’s SELinux user mapping to user_u:

sudo semanage login -a -s user_u regularuser

The change won’t take effect until regularuser logs out and logs back in. If we now run the id -Z command again (or semanage login -l) to see the new SELinux context for regularuser.

NOTA: We can assign a SELinux user to a group user simply writing the group with a first "%" character, like this: sudo semanage login -a -s user_u %regularusers

Let’s see another example of restricting user access through SELinux. By default, SELinux allows users mapped to the guest_t account to execute scripts from their home directories. We can run the getsebool command to check the boolean value (by default is on) like this: getsebool allow_guest_exec_content. If we change the SELinux user mapping for the "regularuser" account executing sudo semanage login -a -s guest_u regularuser, and turn off former boolean executing sudo setsebool allow_guest_exec_content off, we will see "regularuser" can't execute any script from their home folder. So this is how SELinux can apply an additional layer of security on top of DAC. Even when the user has full read, write, execute access to the script created in their own home directory, they can still be stopped from executing it

Let's see another example of restricting user access through SELinux. What if you want to stop this particular user from starting the httpd service even when the user's account is listed in the sudoers file? First, execute sudo semanage login -a -s user_u regularuser. Now that regularuser has been restricted to user_u (and that means to role user_r and domain user_t), we can verify its access executing *sudo semanage user -l* or this other new command:

sudo seinfo -u user_u -x

The output shows the roles user_u can assume (it is user_r). Taking it one step further, we can run the seinfo command to check what domains the user_r role is authorized to enter:

sudo seinfo -r user_r -x

There are a lot of domains user_r is authorized to enter but ...does this list show httpd_t as one of the domains? There are a number of httpd related domains the role has access to, but httpd_t is not one of them. So, if the regularuser account tries to start the httpd daemon, the access should be denied because the httpd process runs within the httpd_t domain and that's not one of the domains the user_r role is authorized to access. And we know user_u (mapped to regularuser) can assume user_r role. This should fail even if the regularuser account has been granted sudo privilege.

Relabeling filesystem:

It will be necessary to relabel the complete filesystem when enabling SELinux after it has been disabled or when changing the SELinux policy from the default targeted policy to strict. To automatically relabel the complete filesystem upon reboot, do:

sudo touch /.autorelabel
sudo reboot

Rebooting may take a while. That's because, in this circumstance, SELinux relabels every last file.

Creating a custom SELinux Policy

If SELinux is denying access for something you believe should be allowed, sealert can provide you a hint how to fix the problem but in more complex cases you'll need to add custom rules to your policy. For that you need these tools:

audit2why : translates SELinux messages into a description of why the access was denied. It's equivalent to execute audit2allow with -w argument.
audit2allow: generates SELinux policy allow/dontaudit rules from logs of denied operations

So you can begin running this command...:

sudo audit2why < /var/log/audit/audit.log
(or *sudo audit2why -a* or *sudo audit2allow -w -a*)

...to show at screen "what's wrong is happening". To see what policy *audit2allow* suggests, type:

sudo audit2allow -a

To implement it (first on screen) you can simply grep for the misbehaving process and pass these resulting AVC logs to *audit2allow* like this:

sudo grep httpd_t /var/log/audit/audit.log | sudo audit2allow -m my_policy_name
(or *sudo grep httpd_t /var/log/audit/audit.log > errors.txt && sudo audit2allow -m my_policy_name <errors.txt*)

Review the result to check if it makes sense (ensure your grep statement does not catch too much). If you're confident its okay, run the same command again with a capital *-M* as parameter: it instructs the command to create a type enforcement (.te) file with the specified name (there you will be able to view the rules to be added) but it also compiles the rule into a SELinux policy package (.pp) which can be installed with the 'semodule' command:

sudo grep httpd_t /var/log/audit/audit.log | sudo audit2allow -M my_policy_name
sudo semodule -i my_policy_name.pp

> NOTA: Do not use the "-M" option to specify the same module name more than once. For example, if you run the command below once with "-M local", and want to run it again later, choose a different name, such as "-M local2".

When an SELinux-enabled system starts, the policy is loaded into memory. SELinux policy comes in modular format, much like the kernel modules loaded at boot time. And just like the kernel modules, they can be dynamically added and removed from memory at run time. The policy store used by SELinux keeps track of the modules that have been loaded. The sestatus command shows the policy store name and the semodule -l command lists the SELinux policy modules currently loaded into memory:

sudo semodule -l | less

> NOTA: 'semodule' can be used for a number other tasks like installing, removing, reloading, upgrading, enabling and disabling SELinux policy modules.

Most modern distributions include binary versions of the modules as part of the SELinux packages and they are located inside "/var/lib/selinux/targeted/active/modules" folder : if you look closely, they will relate to different applications but they're not human-readable, though.

The way SELinux modularization works is that when the system boots, policy modules are combined into what's known as the active policy. This policy is then loaded into memory. The combined binary version of this loaded policy can be found under the "/etc/selinux/targeted/policy"

SELinux sandbox

As the name implies sandbox is used to run any command in a fully user-controllable SELinux-enforced sandbox. This allows you to isolate commands/applications from the rest of your system and only grant specific permissions and capabilities. Even better, SELinux includes a handful of prebuilt sandbox types that allow access to certain critical resources. After you install one dependency (policycoreutils-sandbox), you can even run X apps in an SELinux sandbox! Here's the magic command for running Firefox in an SELinux sandbox on Fedora:

sandbox -X -t sandbox_net_t -t sandbox_web_t -w 1280x1024 firefox

This runs a sandbox with its own X server (-X), allows ports required for web browsing and general network access (-t sandbox_web_t and -t sandbox_net_t) and launches firefox in a 1280x1024 window (-w 1280x1024). This will open up a new window with a completely clean instance of Firefox that is isolated from the rest of your processes by SELinux. Note that this also means you won't be able to access any of your files (including your Firefox profile) so you will get a completely fresh session every time.

If you find this handy, you might want to try extending this to other apps that you use where you might want to test things in clean environments or are handling files you don't entirely trust. For example, here's how to open a report.pdf from my home directory in a sandboxed PDF viewer:

sandbox -X -w 1280x1024 -i ~/report.pdf evince report.pdf

Multilevel security (MLS)

MLS is the fine-grained part of an SELinux security context. It gives the concept of hierarchical "sensitivity". By hierarchy, we mean the levels of sensitivity can go deeper and deeper for more secured content in the file system. Level 0 (depicted by s0) is the lowest sensitivity level, comparable to say, “public.” There can be other sensitivity levels with higher s values: for example, internal, confidential, or regulatory can be depicted by s1, s2, and s3 respectively. This mapping is not stipulated by the policy: system administrators can configure what each sensitivity level mean.

When a SELinux enabled system uses MLS for its policy type (configured in the "/etc/selinux/config" file), it can mark certain files and processes with certain levels of sensitivity. The lowest level is called “current sensitivity” and the highest level is called “clearance sensitivity”.

Going hand-in-hand with sensitivity is the category of the resource, depicted by c. Categories can be considered as labels assigned to a resource. Examples of categories can be department names, customer names, projects etc. The purpose of categorization is to further fine-tune access control. For example, you can mark certain files with confidential sensitivity for users from two different internal departments.

For SELinux security contexts, sensitivity and category work together when a category is implemented. When using a range of sensitivity levels, the format is to show sensitivity levels separated by a hyphen (for example, s0-s2). When using a category, a range is shown with a dot in between. Sensitivity and category values are separated by a colon (:). Here is an example of sensitivity / category pair: "user_u:object_r:etc_t:s0:c0.c2". There is only one sensitivity level here and that’s s0. The category level could also be written as c0-c2.

So where do you assign your category levels? Let’s find the details from the /etc/selinux/targeted/setrans.conf file We won’t go into the details of sensitivities and categories here. Just know that a process is allowed read access to a resource only when its sensitivity and category level is higher than that of the resource (i.e. the process domain dominates the resource type). The process can write to the resource when its sensitivity/category level is less than that of the resource.