SELinux Support in Quattor

Michel Jouvin

jouvin@lal.in2p3.fr

Quattor Workshop, RAL

October 25, 2017

Outline

- Context – Why SELinux?
- What is SELinux
- Customizing SELinux
- Hints for enabling SELinux with Apache
- SELinux and NFS
- SELinux and Quattor
- Useful links
- Conclusions

Context - Disclaimer

- Work referred to in this presentation has been done in the context of migration (virtualized) services, mainly web servers, from EL6 to EL7 (7.3 and 7.4)
  - Not an attempt do use SELinux everywhere
- This work was not started as a planned activity but rather as an attempt to avoid disabling SELinux during VM image build
  - VM image built by applying a Quattor profile to the OpenStack CentOS 7 base image that has SELinux enabled in enforcing mode
- I'm definitely not a SELinux expert but a recent adopter
  - I want to share what I learned…
  - And I'm interested by feedback!
- This presentation contains some early thoughts on how Quattor could help to manage a system with SELinux installed

Why SELinux?

- At LAL, we are among the sites who, since SELinux inception, were waiting to see what was the real future of this product…
  - Disable on all our servers to get rid of the messages on the console!
- But at each OS version, its coverage seems to extend
  - CentOS 7: now a well-documented first-class citizen (https://wiki.centos.org/HowTos/SELinux)
  - OpenStack configures its services to use it by default
- Participate to the “security confinement” of application
  - E.g.: can control with a fine granularity which files and ports an application can access and what they can do with them
  - Potential to reduce the impact of security exploit… as long as a root shell cannot be launched
- Installed Indico v2 that documents SELinux support and it seemed easy…

What is SELinux?

- Linux implementation of a Mandatory Access Control (MAC)
  - MAC = anything not explicitly allowed is forbidden
  - Not only files but also ports and potentially others resources in the system
  - Access right not only based on user identity but also on the application that access the object and the operation the application wants to do
- Differs from Discretionary Access Control (DAC) implemented by permissions
  - An application running as root can do anything
  - A user owning a file can set crazy permissions leading to unnecessary or unsuitable access by applications
  - Rely on the application doing the proper things to protect against misconfigured access or misuse of the application: drop of root privileges, chroot… but not way to enforce it
- SELinux supplements permissions: it never overrides them
  - Permissions are checked first and must allow access to the object

SELinux : Main Concepts

- SELinux security context: an identifier attached to every subject/object
  - Files, processes, ports…
  - Can be displayed with option *–Z* of the object related commands: *ls, ps, id*… + *stat* command for files
  - Format is *user:role:type[:mls]*, e.g. system_u:object_r:httpd_sys_content_t
  - By convention, type names ends with *_t*
- 4 types of access control for the default (and generally used) *targeted* policy
  - By default, only context *type* is checked (Type Enforcement, TE)
  - Role Based Access Control (RBAC): a SELinux *user* and *role* is assigned at login to a user and is checked when accessing objects. *Role* can be used in sudo to restrict what the user can actually do.
  - Multi-Category Security (MCS): use of the *mls* field to implement compartmentalization of VMs and containers

SELinux Policies

- A policy defines what is allowed for a given application
  - Exceptions to the “no access” default
- Can be defined or supplemented with a specific language (in fact several…)

```
cat indico.cil
…
(allow logrotate_t init_t (service (start)))
(allow policykit_t logrotate_t (dbus (send_msg)))
(typetransition unconfined_service_t usr_t sock_file "uwsgi.sock" httpd_sys_rw_content_t)
(filecon "/opt/indico/web/uwsgi\.sock" socket (system_u object_r httpd_sys_rw_content_t ((s0)(s0))))
…
```

- In *targeted* mode, a policy required for every process that has a defined SELinux access type
  - Most standard services (daemons) have a SELinux access type and a standard policy
  - User applications by default are ignored by SELinux (type *unconfined_t*)
  - Rarely have to write a policy in fact…

SELinux Modes

- SELinux can be enabled in either of two modes: *enforcing* and *permissive*
  - enforcing: SELinux enforces MAC, according to the defined policies
  - permissive: SELinux evaluates but doesn't enforce MAC policies
  - In both cases, result of applying the policies are logged into /var/log/audit/audit.log
- Configuring the default mode at boot time : /etc/selinux/config
  - Variable *SELINUX*
  - In addition to the 2 SELinux modes, *disabled*: SELinux completely disabled, *-Z* option disabled in commands, no audit.log. Prefer *permissive*.
- Changing/getting the SELinux mode on a running system: *setenforce* 0 or 1
  - Not possible if SELINUX=disabled
  - *setenforce* is the **only** SELinux command which doesn't survive reboot
  - Getting the current mode: *getenforce* or *sestatus*

File SELinux Context (Label)

- Most services expect files to have a defined SELinux *type (label)* to allow access to them
  - E.g. Apache has a different type for config files, log files, page contents…
  - May need to be explicitly defined/redefined, e.g. after moving a file/directory
  - Commands to manage the SELinux context **ends** with con: *chcon*, *restorecon*
  - To see a file context: *ls –Z* or *stat*
  - *default_file_t* is a default type for files that can be used in policies (*file_t* marks an unlabeled file)
- *restorecon path*: restore the default type for a path, if defined
  - Very handy after relocating a standard file or directory
  - *-R* for recursively resetting the type
- *chcon –t type path*: a chmod-like command to set explicitly the file context
  - By default, follow symlinks. Use *–h* to override.
  - *-R* for recursively setting the type
  - Will be overridden by *restorecon*: prefer *semanage fcontext + restorecon* when possible
  - Example: *chcon –R –t httpd_conf_t /pdisk/httpd/conf*
- Rsync: add option –X to copy the SELinux context (extended attributes)

Customizing SELinux Policies…

- Most policies have booleans that can be used to customize them (*setsebool*)
  - Very useful: default behavior of policies is to prevent anything…
  - E.g. give httpd the right to connect an external DB:

    setsebool [-P] httpd_can_network_connect_db 1
    - -P: make the change permanent (not the default)
  - *getsebool*: show the current value of a given boolean or all Booleans (-a)
- Ports: each service is allowed to access/use only a limited set of ports
  - Basically the standard ports for the service
  - List the SELinux type (service) with access to ports: *semanage port --list*
  - Add access to a port for a service: *semanage port --add*

    semanage port --add -t http_port_t -p udp 82
  - Remove access to a port: *semanage port --delete*

… Customizing SELinux Policies

- Extending a policy, e.g. allow a file to be a symlink or disable auditing of some access errors
  - Requires to create a *type enforcement* policy that will complement the base policy
  - Well described at https://wiki.centos.org/HowTos/SELinux#head-aa437f65e1c7873cddbafd9e9a73bbf9d102c072 but need to understand what you are enabling
    - *audit2allow* can create a policy source file from the errors in audit.log
  - Favor booleans and *semanage port* where possible
- Disable SELinux for a service (SELinux domain)
  - Can be a good approach to workaround a problem without completely disabling SELinux
  - *semanage permissive -a domain*: set the domain (service) in permissive mode
  - *semanage permissive -d domain*: restore enforcing mode for the domain
  - Change is permanent (survive reboots)

Apache (Nginx?) SELinux Hints

- Set the appropriate type for each category of files/directories, in particular if you relocate them
  - By default, symlinks accepted in place of the standard files and directories but both the symlink and the target must have the appropriate type
  - *httpd_conf_t*: an httpd configuration file or a directory containing them
  - *httpd_log_t*: an httpd log file or a directory containing them
  - *httpd_sys_content_t*: a file/directory containing read-only content to serve
  - *httpd_sys_rw_content_t*: a file/directory containing read-write content to serve/update
- Set the appropriate booleans to enable access to external services
  - *httpd_can_network_connect_db*: allow httpd to contact an external database (most backends supported on their standard ports)
  - *httpd_can_sendmail*: allow httpd to send emails

SELinux and NFS

- SELinux allows to define a security context overriding the file context
  - *mount –o context user_t:role_t:type_t* (*system_u:object_r:type*)
  - Used in particular for file systems not supporting extended attrs used to store SELinux labels
- NFS enforces a SELinux context at the file system level
  - Default *nfs_t, c*an be changed at mount time with *–o context*
  - The non default context can also be put in */etc/fstab*
  - See https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7-Beta/html/SELinux_Users_and_Administrators_Guide/sect-Security-Enhanced_Linux-Working_with_SELinux-Mounting_File_Systems.html
- Potential issue with a web server in particular, if you want to support both read-only and read-write areas from the same file system
  - 2 different SELinux types: need to use *httpd_sys_rw_content_t* for the whole file system
  - Anything that I missed?

SELinux and Quattor…

- Most important: preserve the appropriate context for files and directories managed by Quattor
  - Must include symlinks
  - Nothing required if the object managed by Quattor inherits the context from the parent directory
  - Quattor should allow to define an explicit context, define and/or apply a default context
  - Some affected components: ncm-dirperm, ncm-symlink, ncm-filecopy, ncm-metaconfig, ncm-download…
  - but potentially any component creating a file that cannot inherit from its parent!
  - Proposal: SELinux context handling in CAF + schema extension for affected components if needed
    - By default CAF should do a *restorecon* if a default context is defined for the file/dir/symlink

… SELinux and Quattor

- Managing SELinux itself: ncm-selinux?
  - Manage permanent/current SELinux mode: /etc/selinux/config + setenforce
    - Could take the ncm-chkconfig approach with defining the permanent service state + startstop
  - Manage booleans: probably the most important
  - Manage ports: useful too
  - Setting a domain in permissive mode: probably not, should remain a workaround rather than a config option
  - Manage policies: looks really complex but could activate a policy that has been downloaded with ncm-download
    - Can it be considered secure enough?
    - A broken policy will prevent the system to reboot

Useful Links and Documentation

- CentOS documentation: https://wiki.centos.org/HowTos/SELinux
- RedHat: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7-Beta/html/SELinux_Users_and_Administrators_Guide/index.html
  - Different from CentOS documentation
- Package selinux-policy-doc to get the man documentation of the various standard policies
  - Can be found on the web too… if you know the name of the man page!
  - No general index unfortunately but *apropos* can help: http://www.unix.com/apropos-man/centos/8/selinux/
- StackOverflow entries are often quite voodoo: don't follow them if you are sure to understand them!

Conclusions

- Started to look at SELinux by curiosity… but turned out to be not so difficult to get it configured and enabled on some productions services
  - Mainly Apache-based web servers running various applications, including uWSGI applications
  - Also an OpenLDAP server, a cups server…
- Implementing Mandatory Access Control: fine-grained control on who can do what
  - E.g. can prevent a compromised httpd server to write files everywhere
  - Doesn’t protect against an exploit allowing to start a shell as root
- New concepts: require to take the time to understand it
  - A lot of good documentation available on the web: prefer the official ones!
- Use permissive mode if you cannot afford to enable enforcing mode
  - audit.log will help to understand what has to be done
- Quattor: a minimal support could probably be implemented in CAF for files, directories, symlinks…