System Security
Capabilities / credentials

Samuel Thibault <samuel.thibault@u-bordeaux.fr>
https://dept-info.labri.fr/~thibault/enseignements#SecuSys

PAM

Linux Pluggable Authentication Modules

- Centralize authentication questions

Services

- login
- sshd
- DM

Modules

- unix
- kerberos
- ldap
- fingerprint reader
- ...

PAM

4 types of module features :

- account
  - expiration, time of day, ...
- authentication
  - password, token, ...
- password
  - updating the password
- session
  - tuning the session
    - welcome banner
    - hardware access
    - memory quotas

POSIX Capabilities

root / non-root is too binary

setuid is awful

For instance, ping :

- On old distributions (e.g. Debian 9 stretch)

```
$ ls -l /bin/ping
-rwsr-xr-x 1 root root 61240 Nov 10  2016 /bin/ping
```

- Just because it needs to send raw packets over the network
- On newer distributions (e.g. Debian 10 buster)

```
$ ls -l /bin/ping
-rwxr-xr-x 1 root root 76K  2 févr.  2021 /bin/ping
# getcap /bin/ping
/bin/ping cap_net_raw=ep
```

- This is a POSIX *capability*

POSIX Capabilities

POSIX capability : a precise administration right, e.g. :

- CAP_CHOWN
- CAP_KILL
- CAP_NET_ADMIN
- CAP_NET_RAW
- CAP_SYS_NICE
- ... (see **`man 7 capabilities`**)

- Processes have a list of capabilities (in addition to uid/gids)
- Program binaries can have a list of capabilities
  - Similar to setuid

→ Allows to fine-tune administration rights delegation

POSIX Capabilities

They are stored in the FS as *extended attributes (xattr)*

```
$ xattr /bin/ping
security.capability
$ xattr -p security.capability /bin/ping |
hexdump
01 00 00 02 00 20 00 00  00 00 00 00 00 00 00 00
00 00 00 00 0a
```

non-POSIX Capabilities

E.g. Capsicum

Capabilities attached to an opened file

- CAP_READ
- CAP_SEEK
- CAP_MMAP_W
- ...

Limit the system calls that one can use of it

- Could seem redundant with seccomp+bpf
- But on a real capability-based OS, one can transfer them between programs

Passing credentials / capabilities over

```
man 7 unix
man 7 cmsg
```

Local UNIX sockets let the kernel make a direct relation between two processes

- Identify the other end
  - (identity when the socket was created)
  - SO_PEERCRED
  - SO_PEERSEC
- Pass a chosen identity
  - SO_PASSCRED, SCM_CREDENTIALS cmsg
  - SO_PASSSEC, SCM_SECURITY cmsg
- Pass a file descriptor
  - SCM_RIGHTS cmsg

More generally, Access Control

Two main models

DAC (Discretionary Access Control)

- Transferrable capabilities
- e.g. file access rights

MAC (Mandatory Access Control)

- Non-transferrable capabilities
- e.g. CPU time, disk space, memory space

LSM (Linux Security Modules)

Main goal : support Mandatory Access Control

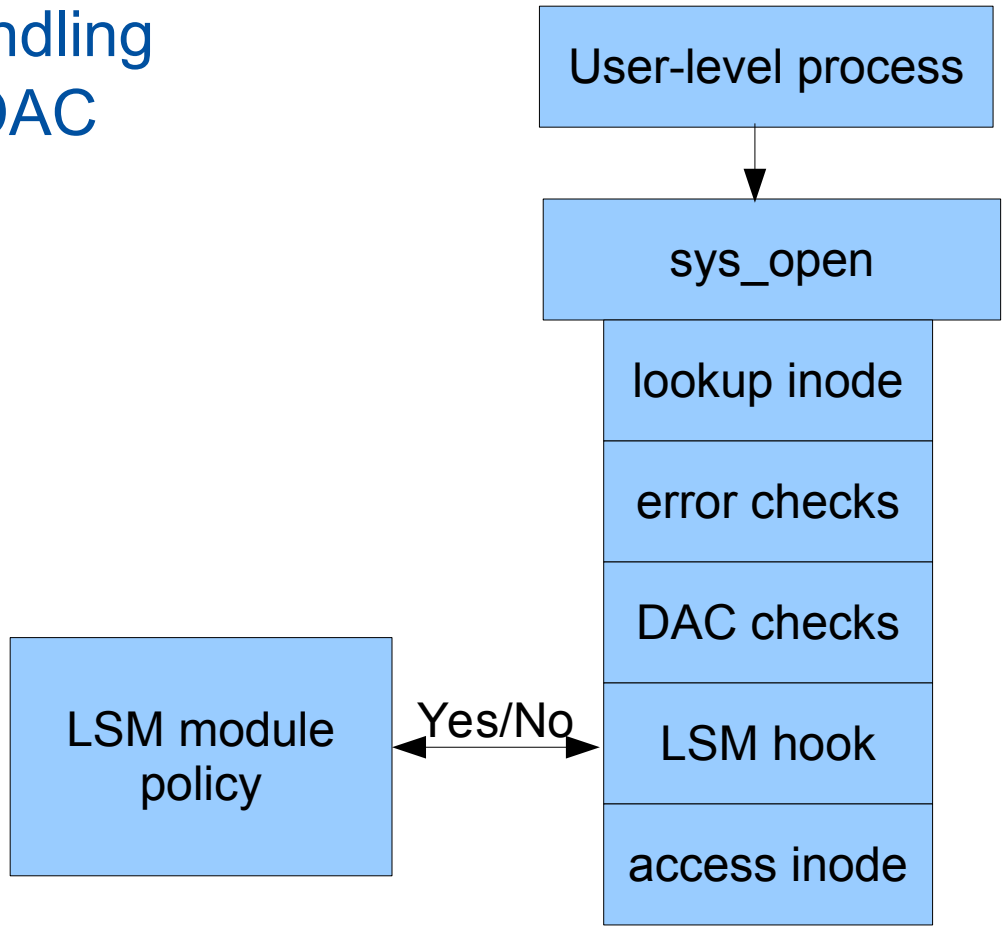
LSM is the kernel hooks support
Then various implementations

- SELinux
- AppArmor
- SMACK
- Tomoyo
- ...

LSM hooks

Plugged in lots of system calls

- Plugged after normal error handling
- Most often plugged after the DAC
  - To be able to contradict it



LSM hooks

TODO : show **`vfs_mkdir`**

« May a subject S perform a kernel operation OP on an internal kernel object OBJ ? »

See linux/include/linux/security.h to see all calls

- security_inode_create/mkdir/rmdir/rename/...
- security_file_ioctl/lock/fcntl/...
- security_task_setnice/setioprio/setrlimit/kill/...
- ...

LSM modules

Most often, configurable list of authorizations, stored :

- In configuration files
- In a database
- In the FS itself (as xattr)

Most often, have a learning mode

- Access violations are logged
- But they are not denied
- Useful for development to track what authorizations are missing
- Disabled on the production system

SELinux

(From the NSA, notably)

- Execution domains on processes, files, programs
  - Stored in the file inodes with xattr
- Rules between these domains

For instance, an apache and a mariadb domain

- Programs in the apache domain
  - can access files in the apache domain
- Programs in the mariadb domain
  - can access the database in the mariadb domain
- But no crossover

Looks like unix users ? More fine-grain

Fine-grain SELinux

The **passwd** command allows users to change their password

- On standard UNIX, has to write to **/etc/shadow**
  - I.e. **/usr/bin/passwd** is setuid, eww
- On SELinux :

```
$ ls -lZ /usr/bin/passwd
-rwsr-xr-x  root root
system_u:object_r:passwd_exec_t:s0 /usr/bin/passwd
$ ls -lZ /etc/shadow
-r--------. root root
system_u:object_r:shadow_t:s0      /etc/shadow
$ passwd
Changing password for user user_name. [...]

$ ps -eZ | grep passwd
unconfined_u:unconfined_r:passwd_t:s0-s0:c0.c1023
13212 pts/1 00:00:00 passwd
```

Fine-grain SELinux

A few rules

- passwd_exec_t makes execution enter passwd_t domain
- **`shadow_t`** files can be written to by the passwd_t domain
- other domains do not have the right to use passwd_exec_t
  - e.g. crond_t, apache_t, bind_t, etc.

AppArmor

Based on configuration files

- Simpler to configure
- Authorizes some programs to do some actions

- Files identified by path, not inode
- « /usr/bin/passwd is allowed to write into /etc/passwd »

Enabled by default in Debian since Buster

And also... PolicyKit

- Completely userland
  - Use credential passing to identify processes
- Lets unprivileged processes discuss with privileged processes
- Authorization depending on configurable rules

- For instance, access to sound rendering daemon (pulseaudio, pipewire)