MUNI
FI

Access Control Mechanisms

PA211 Advanced Topics of Cyber Security

November 15, 2022

Petr Velan, Pavel Čeleda and Jan Vykopal

Agenda

- Response to the Exit Ticket from Last Week
- Discretionary Access Control
  - File and directory permissions
- ACL and Shared Directory Management
- Mandatory Access Control
  - AppArmor
  - SELinux

Exit Tickets from Last Week – I

- **Why did we allow outbound connections on all interfaces instead of concrete ports (22, 80, 443, 2222)?**
- A: Firstly, we have followed CIS Benchmark, which specifies it in this way. Second, you want to limit what is coming INTO your network (host), not out of it. Therefore, everything can go OUT and only something (on port 222, 80, 443, 2222) can go even IN.

Exit Tickets from Last Week – II

- **You said that we should not trust scripts we find on the internet. How do you then evaluate whether the source code is trustworthy or not?**
- A: As always, there is no 100% solution.
- Ideally, you can read source code and understand what's going on
- You trust the source who recommended the tool (friend, teacher, ...) or check evaluation metrics such as GitLab/GitHub stars, the source website rating (StackOverflow score)
- Run in VM for test, however, still insecure – can recognize virtual environment, does not have to be harmful during your test but later can be, ….

Discretionary Access Control

Ownership of Files and Directories

- Each file/directory belongs to a **user** and a **group**
- When a user creates a file/directory, it belongs to him and his **primary** group (an exception is mentioned later)
- Owner can change the group of a file/directory to any of his secondary groups using `gchrp` or `chown` commands without `sudo`
- Changing the owner of file always requires root privileges

MUNI FI

Permissions of Files and Directories

- Each file/directory has three basic permissions
  - r: Read permission
  - w: Write permission
  - x: Execute permission for files, access permission for directories
- A set of permissions is applied separately to user, group, and others.

```
$ ls -la example
total 8
drwxr-xr-x 2 vagrant test    4096 Nov 13 22:36 .
drwxr-xr-x 7 vagrant vagrant 4096 Nov 13 22:52 ..
 usrgrpoth
```


MUNI FI

Symbolical vs Numerical Representation

- Permissions can be represented symbolically or numerically
- r (4), w (2), x (1) => r-x (5), rw- (7), …

```
$ stat example
  File: example
  Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: 801h/2049d      Inode: 991276      Links: 2
Access: (0755/drwxr-xr-x)  Uid: (  900/ vagrant)   Gid: ( 1002/    test)
Access: 2022-11-13 22:53:26.103004503 +0100
Modify: 2022-11-13 22:36:50.103004503 +0100
Change: 2022-11-13 22:52:57.755004503 +0100
Birth: -
```

- Why is there **0755** instead of just **755**?

SUID and SGID on Files

- SUID (setuid) bit causes **effective** user ID to be set to the owner of the accessed (executable) file
- SGID (setgid) bit causes **effective** group ID to be set to the group of the accessed (executable) file

```
$ ll /usr/bin/ssh-agent /usr/bin/sudo
-rwxr-sr-x 1 root ssh  321672 Jan 31  2020 /usr/bin/ssh-agent
-rwsr-xr-x 1 root root 157192 Jan 20  2021 /usr/bin/sudo
```

- When the executable bit is not set, the **s** changes to **S**
- Numerical value for SUID is 4000, for SGID 2000

Process User/Group IDs

- *Effective ID*: the ID that is checked when permissions are verified
- *Real ID*: the ID of the user/group that started the process
- *Saved ID*: used e.g. when SUID program needs to do unprivileged work (both real and effective IDs are that of unprivileged user)
- Normal processes start with all IDs belonging to user and primary group of the user that started the process

SUID/SGID Security Concerns

- Executables with SUID/SGID permissions are critical
- Any vulnerability potentially leads to privilege escalation
- CVE-2021-3156 (Sudo Heap-Based Buffer Overflow)
  - https://github.com/0xdevil/CVE-2021-3156
  - Easy way into older unpatched Linux machines
- Attacker can leave SUID/SGID binaries as a backdoor

SUID/SGID Mitigation

- Search and eradicate
  - `sudo find / -type f \( -perm -4000 -o -perm -2000 \) -exec ls -l --color {} \;`
  - `-type f`: search for files
  - `\( ... -o ... \)`: two search terms with *or*
  - `-exec ... {} \;`: execute a command for each found file
- Mount partition with nosuid or noexec options
  - nosuid: SUID and GUID permissions are ignored on the mount point
  - noexec: forbids execution of any program on the mount point

umask

- User file creation mode mask

```
$ umask
0022
$ touch example
$ ls -l example
-rw-r--r-- 1 vagrant vagrant 0 Nov 14 00:00 example
$ umask 066
$ touch example2
$ ls -l example2
-rw------- 1 vagrant vagrant 0 Nov 14 00:01 example2
```

- Important for secure file creation
- umask of any process can be found in `/proc/[pid]/status`
- If default ACL exists, it replaces umask

MUNI FI

Extended File Attributes 1/2

- Set of attributes defining behavior of the file
- Not all attributes are implemented in each filesystem
- List extended attributes: `lsattr(1)`
- Change extended attributes: `chattr(1)`, requires `sudo`

```
$ lsattr example
----ia--------e---- example
$ lsattr -l example
example                        Immutable, Append_Only, Extents
```


MUNI FI

Extended File Attributes 2/2

- **append only (a)**
- no atime updates (A)
- compressed (c)
- no copy on write (C)
- no dump (d)
- synchronous directory updates (D)
- extent format (e)
- **immutable (i)**
- data journalling (j)
- project hierarchy (P)
- secure deletion (s)
- synchronous updates (S)
- no tail-merging (t)
- top of directory hierarchy (T)
- undeletable (u)

ACL and Shared Directory Management

Access Control List (ACL)

- ACL allows us to set more granular permissions
  - Individual permissions for users and groups
  - See acl(5)
- Supported by most common file systems
  - `Ext2, Ext3, Ext4, ReiserFS, Btrtr, XFS, …`
- `getfacl(1)`: get file access control lists
- `setfact(1)`: set file access control lists
  - -m: modify ACL entries
  - -x: remove ACL entries
  - Permissions can be set for [u]ser, [g]roup, [o]ther , and [m]ask

ACL Example

```
$ echo example > example

$ ls -l example
-rw------- 1 vagrant vagrant 8 Nov 14 01:31 example

$ getfacl example
# file: example
# owner: vagrant
# group: vagrant
user::rw-
group::---
other::---
```

```
$ setfacl -m g:intern2:rw example
$ ls -l example
-rw-rw----+ 1 vagrant vagrant 8 Nov 14 01:55 example

$ getfacl example
# file: example
# owner: vagrant
# group: vagrant
user::rw-
group::---
group:intern2:rw-
mask::rw-
other::---

$ sudo su intern2
intern2$ echo write >> example
intern2$ cat example
example
write
```


MUNI FI

ACL Mask

– A mask ACL entry specifies the maximum access which can be granted by any ACL entry except the user entry for the file owner and the other entry

```
$ setfacl -m m:r example

$ getfacl example
# file: example
# owner: vagrant
# group: vagrant
user::rw-
group::---
group:intern2:rw-               #effective:r--
mask::r--
other::---
```


MUNI FI

Default ACL

- Applies to a directory
- Each new object in the directory inherits the ACL
- Works recursively

```
$ mkdir example
$ setfacl -m d:g:intern2:rw example
$ mkdir example/example2/
```

```
$ getfacl example/example2/
# file: example/example2/
# owner: vagrant
# group: vagrant
user::rwx
group::r-x
group:intern2:rw-
mask::rwx
other::r-x
default:user::rwx
default:group::r-x
default:group:intern2:rw-
default:mask::rwx
default:other::r-x
```

Archiving with ACL

- How to preserve ACL when archiving or copying files?
- `tar -acl`
- `mv` keeps ACL intact
- `cp --preserve`: preserves ACL

Shared Directory

- Allow multiple users to access and modify each other's files
  - Use same group
  - Set SGID on directory – each new object inherits the same group
  - If users are not to delete each other's file, set **sticky bit** (1000)

```
$ mkdir example
$ chgrp test example/
$ chmod 3775 example/
$ sudo usermod -G test -a intern2
$ sudo su intern2
intern2$ touch example/example.txt
$ ls -la example
total 8
drwxrwsr-t 2 vagrant test    4096 Nov 14 02:15 .
drwxr-xr-x 6 vagrant vagrant 4096 Nov 14 02:12 ..
-rw-r--r-- 1 intern2 test       0 Nov 14 02:15 example.txt
```

Mandatory Access Control

Mandatory Access Control

- Limits abilities of processes to:
  - Access, create, delete files
  - Execute other processes
  - Communicate over network
  - Allocate resources

- Two main mechanisms:
  - AppArmor
  - SELinux

AppArmor

- Debian, Ubuntu, and SUSE Linux families
- AppArmor profiles limit processes based on their paths
- Only processes with existing (loaded) profiles are protected
- See `man 7 apparmor`
- Sources:
  - https://medium.com/p/64d7ae211ed
  - https://wiki.debian.org/AppArmor/HowToUse
  - https://ubuntu.com/tutorials/beginning-apparmor-profile-development
  - https://documentation.suse.com/sles/15-SP1/html/SLES-all/part-apparmor.html

AppArmor Modes

- **Enforced:** Profiles loaded in enforcement mode will result in enforcement of the policy defined in the profile as well as reporting policy violation attempts to syslogd.
- **Complain:** Profiles loaded in "complain" mode will not enforce policy. Instead, it will report policy violation attempts. Note that deny rules in profiles are enforced/blocked even in complain mode.
- **Unconfined:** No profile is loaded for the process

AppArmor Profiles

- Profiles are applied during start of the process (`exec`)
- Stored in `/etc/apparmor.d/`
- Abstractions: parts of profiles that can be reused
- Tunables: variables to be used in profiles
- Most profiles support extension by including from `/etc/apparmor.d/local`
- Reapply profile after a change:

```
apparmor_parser -r /path/to/profile
```

- Profiles for most common applications are packaged in `apparmor-profiles` and `apparmor-profiles-extra`

AppArmor Profile Example

```
$ cat /etc/apparmor.d/bin.ping
...

#include <tunables/global>
profile ping /{usr/,}bin/{,iputils-}ping flags=(complain) {
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>

  capability net_raw,
  capability setuid,
  network inet raw,
  network inet6 raw,

  /{,usr/}bin/{,iputils-}ping mixr,
  /etc/modules.conf r,

  # Site-specific additions and overrides. See local/README for details.
  #include <local/bin.ping>
}
```

AppArmor Commandline Utilities

- Package apparmor
  - aa-status
  - apparmor_parser
  - abstractions, tunables
- Package apparmor-utils
  - aa-enforce
  - aa-disable
  - aa-complain
  - aa-genprof
  - ...

AppArmor Status

- `sudo cat /sys/kernel/security/apparmor/profiles`
- `sudo aa-status`

```
$ sudo aa-status
apparmor module is loaded.
30 profiles are loaded.
14 profiles are in enforce mode.
   /usr/bin/man
   ping
...
1 processes have profiles defined.
1 processes are in enforce mode.
   /usr/sbin/sshd (1194) docker-default
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

Creation of AppArmor Profile

- `aa-genprof`: generate profile for the first time
- `aa-logprof, aa-mergeprof`: update existing profile

```
$ cat example/example.sh
#!/bin/bash

echo "This is an apparmor
example."

touch sample.txt
echo "File created"

rm sample.txt
echo "File deleted"

$ ./example.sh
This is an apparmor example.
File created
File deleted
```

```
$ sudo aa-genprof ./example.sh
…
# in another terminal, run the ./example.sh
$ ./example.sh
# go back to the first one and select Scan system log for AppArmor events
# use Inherit option for rm and touch.

#include <tunables/global>
/home/vagrant/example/example.sh {
  #include <abstractions/base>
  #include <abstractions/bash>
  #include <abstractions/consoles>
  /home/vagrant/example/example.sh r,
  /usr/bin/bash ix,
  /usr/bin/rm mrix,
  /usr/bin/touch mrix,
  owner /home/*/example/sample.txt w,
}
```

AppArmor Debugging

- Log file:
  - `/var/log/audit.log` - if auditd is installed
  - `/var/log/syslog` - without auditd
  - `journalctl`
- By default, operations that trigger "deny" rules are not logged. This is called deny audit quieting.
  - Turn on: `echo -n noquiet >/sys/module/apparmor/parameters/audit`
  - Add audit keyword to profile, e.g.: `audit owner /home/*/.ssh/**        rw,`
- Example:

```
Nov 14 16:10:49 server kernel: [65014.150055] audit: type=1400 audit(1668438649.775:197): apparmor="DENIED"
operation="open" profile="/home/vagrant/example/example.sh" name="/usr/bin/rm" pid=32517 comm="example.sh"
requested_mask="r" denied_mask="r" fsuid=900 ouid=0
```


MUNI FI

AppArmor and Docker

- Docker uses AppArmor to limit containers by default
- Not present in `/etc/apparmor.d/`
- Limits access to `/proc/`, `/sys/`, `mount`, …
- Example:

```
$ sudo aa-profile
...
1 processes are in enforce mode.
   /usr/sbin/sshd (17036) docker-default
...
```


MUNI FI

SELinux (Bonus Content)

- Red Hat Enterprise Linux
- Works by assigning security context to files, directories, and processes
- Needs support from filesystem to store the context
- `ls -Z, ps -Z`
- `getenforce / setenforce` (Enforcing vs Permissive)

SELinux Contexts

```
$ ls -la -Z .ssh
total 12
drwx------. 2 csirt csirt unconfined_u:object_r:ssh_home_t:s0        48 Nov 14 15:50 .
drwx------. 7 csirt csirt unconfined_u:object_r:user_home_dir_t:s0  206 Nov  7 21:50 ..
-rw-r--r--. 1 csirt csirt unconfined_u:object_r:ssh_home_t:s0      5612 Nov 14 15:50 authorized_keys
-rw-r--r--. 1 csirt csirt unconfined_u:object_r:ssh_home_t:s0       936 Mar  4  2022 known_hosts
```

```
$ ps aux -Z | grep sshd
system_u:system_r:sshd_t:s0-s0:c0.c1023 root 2115 0.0  0.0 94372 7636 ?        Ss   Aug09   0:17
/usr/sbin/sshd -D
```

- Context – user, role, type
- SELinux profile define relationships between process and file types
- Can limit access to individual ports as well
- Manipulating contexts: `chcon, restorecon, semanage`

MUNI
FI