ADS - System administration
Prof. Fabien Dutoit
BA6 - 2023-2024

Lab report #6

Access control

Germano Thomas **Martins Alexis** **Saez Pablo**

04.05.2024

Table des matières

1 - Introduction

It's crucial nowadays to correctly manage and understand how access control works. If one resource is poorly managed, first it's terrible for the user experience, but the security issues that can come from that can be terrible. In this laboratory, we explore the base principals of access control on UNIX systems and also create scripts for real world issues.

2 - Task 1: Exercises

2.1 - Interpreting account and group information

```
$ id
uid=1000(alexis) gid=1000(alexis) groups=1000(alexis),4(adm),24(cdrom),27(sudo),
30(dip),46(plugdev),122(lpadmin),135(lxd),136(sambashare),999(docker)
```

- **What is your UID and what is your account name?** The UID is `1000`, and the account name is `alexis`
- **What is the GID of your primary group ("groupe principal") and what is its name?** The GID of the main group is also `1000`, and the name is also `alexis`. This is because each user has by default a group with it's own name.
- **How many other groups are you a member of?** On my personal computer, I am also members of 9 different groups.

2.2 - Interpreting access control metadata on files and directories

1. For the following files, determine who is the owner, which group owns the file and characterize the group of people who can read, who can write and who can execute the file.

```
$ ls -lisa /etc/passwd
21761272 4 -rw-r--r-- 1 root root 2937 Mar  2 23:26 /etc/passwd
$ ls -lisa /bin/ls
27788500 136 -rwxr-xr-x 1 root root 138216 Feb  8 04:46 /bin/ls
$ ls -lisa ~/.bashrc
9043971 4 -rw-r--r-- 1 alexis alexis 3792 Feb  9 21:38 /home/alexis/.bashrc
$ ls -lisa ~/.bash_history
9044927 4 -rw------- 1 alexis alexis 3818 Apr  3 11:03 /home/alexis/.bash_history
```

/etc/passwd
- **Owner**: root
- **Group**: root
- **Owner permissions**: read and write
- **Group permissions**: read
- **Others permissions**: read

/bin/ls
- **Owner**: root
- **Group**: root
- **Owner permissions**: read, write, and execute
- **Group permissions**: read and execute
- **Others permissions**: read and execute

/.bashrc
- **Owner**: alexis
- **Group**: alexis
- **Owner permissions**: read and write
- **Group permissions**: read
- **Others permissions**: read

/.bash_history
- **Owner**: alexis
- **Group**: alexis
- **Owner permissions**: read and write
- **Group permissions**: none
- **Others permissions**: none

2. Examine the permissions of your home directory (what option do you have to pass to ls to examine the permissions of directories?).

```
$ ls -ld ~
drwxr-x--- 40 alexis alexis 4096 May  4 15:39 /home/alexis
```

We have to pass the `-d` option if we want to apply the `ls` on the directory.

/home/alexis
- **Owner**: alexis
- **Owning Group**: alexis
- **Owner Permissions**: read, write, and execute
- **Group Permissions**: read and execute
- **Others Permissions**: none
- **Can List Files**: owner and group
- **Can Create Files**: owner

3. What permissions allow you to create files in the /tmp directory?

```
$ ls -ld /tmp
drwxrwxrwt 23 root root 12288 May  4 15:52 /tmp
```

We need `Write` and `Execute` permission to create files in `/tmp`.

/tmp
- **Owner**: root
- **Owning Group**: root
- **Permissions**: rwxrwxrwt
- **Can Create Files**: everyone with write and execute permissions

2.3 - Modifying access rights

1. Using chmod in symbolic mode, create the following configurations (from initial configuration "600")

```
chmod u=rw,g=r,o=   file  # rw- r-- ---
chmod u=rwx,g=rx,o= file  # rwx r-x ---
chmod u=r,g=r,o=r   file  # r-- r-- r--
chmod u=rwx,g=r,o=r file  # rwx r-- r--
chmod u=rwx,g=,o=   file  # rwx --- ---
```

2. Conflicting permissions, What does the OS do if you try to write to this file?

```
$ touch conflicting_file
$ chmod 444 conflicting_file
$ chmod o+w conflicting_file
$ echo "Hello, world" > conflicting_file
zsh: permission denied: conflicting_file
```

It was expected, but we have a permission error.

2.4 - Giving other users access to your files

1. Is your colleague able to read the files in your home directory? If yes, why? If no, why not?

According to the print of the privileges, the group “other” where are all the other users other than “labX”. The “other” group has read and execute privileges, so the group “other” can list and read the files in our home folder.

```
labh@ads:/home$ ls -lisa
total 52
1048577 4 drwxr-xr-x 13 root     root     4096 Mar 18 11:21 .
      2 4 drwxr-xr-x 19 root     root     4096 Aug 22  2023 ..
1048578 4 drwxr-xr-x  5 heiguser heiguser 4096 May  6 16:07 heiguser
1048594 4 drwxr-xr-x  6 lab0     lab0     4096 Mar 18 11:29 lab0
1048723 4 drwxr-xr-x 12 laba     laba     4096 May  6 15:34 laba
1048852 4 drwxr-xr-x  7 labb     labb     4096 Apr 14 17:17 labb
1048982 4 drwxr-xr-x  6 labc     labc     4096 Apr 14 10:27 labc
1049112 4 drwxr-xr-x  8 labd     labd     4096 Apr 12 11:27 labd
1049241 4 drwxr-xr-x  7 labe     labe     4096 Apr 14 22:32 labe
1049370 4 drwxr-xr-x  7 labf     labf     4096 Apr 29 16:57 labf
1049499 4 drwxr-xr-x  6 labg     labg     4096 Apr 14 22:00 labg
1049629 4 drwxr-xr-x  6 labh     labh     4096 Apr 13 17:32 labh
1049758 4 drwxr-xr-x  7 labi     labi     4096 Apr 14 09:08 labi
```

2. What do you need to do so that your colleague (and maybe others) can read your files? What do you need to do so that nobody else can read your files?

If we want to give them access, we can use the command.

```
chmod go+rx /home/labh/
chmod go+r /home/labh/*
```

If we want to prevent them from reading our files we can do the commands below. It will allow only us to have access to the content of this folder.

```
chmod go-rx /home/labh/
chmod go-r /home/labh/*
```

3. In your home directory create a directory named shared . By using the commands chmod and chgrp configure the directory in such a way that only your colleague is able to read the directory and its files. You can use the groups proj_a and proj_b . Both you and your colleague are already a member of these groups. (For the sake of this exercise, suppose that nobody else is member of this group, only you and your colleague.) Give your colleague also access rights to create new files and modify existing files. What commands did you use ?

```
mkdir /home/labh/shared
chgrp proj_a /home/labh/shared
chmod o-rwx /home/labh/shared
chmod g+rwx /home/labh/shared
```

2.5 - Find

1. What does find do with hidden files or directories?

It shows all the files including the hidden files.

```
$ find .
.
./fileB
./.fileA
```

2. Using find display all the files in your home directory

- that end in .c , .cpp or in .sh

```
$ find ~/ -type f \( -name "*.c" -o -name "*.cpp" -o -name "*.sh" \)
```

- that are executable

```
$ find ~/ -type f -executable
```

- that have not been modified since more than two years

```
$ find ~/ -type f -mtime +730
```

- that have not been accessed since more than two years

```
$ find ~/ -type f -atime +730
```

- that have not been accessed since more than three years and that are bigger than 3 MB (good candidates for cleanup)

```
$ find ~/ -type f -atime +1095 -size +3M
```

- Display directories called .git on your home directory

```
$ find ~/ -type d -name ".git"
```

3. Suppose your current directory has no subdirectories. You want to display all files that contain the word root . Which of the two commands is correct and why?

The correct command is the one below (with the “.”). Because the command also look for the hidden files, and the command with the “*” ignores the hidden files.

```
$ find . -type f -exec grep -l 'root' {} \;
```

Here is an example of what I mean with the explication above.

```
$ find . -type f -exec grep -l 'root' {} \;
./.test3 --> Hidden file is here
./test2
t$ find * -type f -exec grep -l 'root' {} \;
test2
```

3 - Task 2: Display world-writable files

Write a script named fix_permissions that for the time being only displays the worldwritable files and directories.

```bash
#!/bin/bash

# Display the world-writable files and directories
echo "The following files/directories are world-writable:"
find test_dir -perm -o+w
```

4 - Task 3: Pass the directory as an argument

The user should be able to specify which directory the tool should analyse. Modify the script such that it takes one argument, the directory. The script should behave as follows:

- If it is called without argument it should display a corresponding error message and exit with a return code 1.
- If the given argument is not a valid directory it should display a corresponding error message and exit with a return code 1.

```bash
#!/bin/bash

# Check if an argument was provided
if [ -z "$1" ]; then
    echo "Error: No directory specified."
    exit 1
fi

# Check if the provided argument is a valid directory
if [ ! -d "$1" ]; then
    echo "Error: '$1' is not a valid directory."
    exit 1
fi

# Display the world-writable files and directories
echo "The following files/directories are world-writable:"
find "$1" -perm -o+w
```

5 - Task 4: Propose a fix

The tool should not only diagnose problems, but also offer a fix, that is, remove the offending permissions from the files. But the tool should not do that automatically, it should first ask the user if he wants to do this.

```bash
#!/bin/bash

# Check if an argument was provided
if [ -z "$1" ]; then
    echo "Error: No directory specified."
    exit 1
fi

# Check if the provided argument is a valid directory
if [ ! -d "$1" ]; then
    echo "Error: '$1' is not a valid directory."
    exit 1
fi

# Find world-writable files and directories
world_writable=$(find "$1" -perm -o+w)

# Check if any world-writable files or directories were found
if [ -z "$world_writable" ]; then
    echo "No world-writable files or directories found."
    exit 0
else
    echo "The following files/directories are world-writable:"
    echo "$world_writable"
fi

# Ask the user if they want to fix the permissions
read -p "Do you want the permissions to be fixed (y/n)? " answer

if [[ "$answer" =~ ^[Yy]$ ]]; then
    # Fix the permissions by removing write permission for others
    find "$1" -perm -o+w -exec chmod o-w {} \;
    echo "Permissions have been fixed."
else
    echo "Permissions have not been modified."
fi
```

6 - Task 5: Display group-writable files

```bash
#!/bin/bash

# Check if an argument was provided
if [ -z "$1" ]; then
    echo "Error: No directory specified."
    exit 1
fi

# Check if the provided argument is a valid directory
if [ ! -d "$1" ]; then
    echo "Error: '$1' is not a valid directory."
    exit 1
fi

# Get the username and personal group
USERNAME="$USER"
PERSONAL_GROUP=$(id -gn "$USERNAME")

# Find group-writable files and directories that are not owned by the personal group
group_writable=$(find "$1" -perm -g+w ! -group "$PERSONAL_GROUP")

# Check if any group-writable files or directories were found
if [ -z "$group_writable" ]; then
    echo "No group-writable files or directories found."
else
    echo "The following files/directories are writable for groups:"
    echo "$group_writable"
fi
```

7 - Conclusion

The first part of the lab really allowed us to start practicing all the access control mechanisms/principals. Meanwhile in the second part, we had a real case scenario which required to create some scripts to answer the problematic. This last part allowed us to really master what we were doing. Moreover as a group composed of cybersecurity students, we enjoyed look at a part important for our jobs.