Lab6 - Access control

ADS

Felix Breval, Anthony David, Timothée Van Hove

May 12, 2024

Contents

Task 1: Exercises

Interpreting account and group information

Execute the command `id` to display information about your account and the groups you are member of.

- What is your UID and what is your account name?
- What is the GID of your primary group ("groupe principal") and what is its name?
- How many other groups are you a member of?

Output :

```
$ id
uid=1000(anthony) gid=1000(anthony)
↪  groups=1000(anthony),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),122(lpadmin),133(lxd),134(sambashare)
```

Questions :

- The UID is `1000` and the account name is `anthony`.
- The GID is `1000` the the name of the primary group is `anthony`.
- I am a member of 8 others groups (`adm`, `cdrom`, `sudo`, `dip`, `plugdev`, `lpadmin`, `lxd`, `sambashare`).

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.

- `/etc/passwd`
- `/bin/ls`
- `~/.bashrc`
- `~/.bash_history`

Input and Output :

`/etc/passwd`

```
$ ls -l /etc/passwd
-rw-r--r-- 1 root root 2727 Feb 23 19:06 /etc/passwd
```

- `-l`: Displays detailed information in a long listing format.

`/bin/ls`

```
$ ls -l /bin/ls
-rwxr-xr-x 1 root root 138216 Feb  8 04:46 /bin/ls
```

`~/.bashrc`

```
$ ls -l ~/.bashrc
-rw-r--r-- 1 anthony anthony 3792 Okt 20  2023 /home/anthony/.bashrc
```

‘~/.bash_history

```
$ ls -l ~/.bash_history
-rw------- 1 anthony anthony 14225 Dez  8 17:28 /home/anthony/.bash_history
```

File ownership and permission details :

File	Owner	Group	Can Read	Can Write	Can Execute
`/etc/passwd`	root	root	Everyone	Only root	No one
`/bin/ls`	root	root	Everyone	Only root	Everyone
`~/.bashrc`	anthony	anthony	Everyone	Only anthony	No one
`~/.bash_history`	anthony	anthony	Only anthony	Only anthony	No one

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

- Who is the owner and which is the owning group?
- What is the configuration of permissions?
  - Who can list files?
  - Who can create files?

Input and output :

```
$ ls -ld ~
drwxr-x--- 17 anthony anthony 4096 mai    6 17:19 /home/anthony
```

- `-l`: Displays detailed information in a long listing format.
- `-d`: Lists information about the directory itself, not its contents.
- `rwx` for the owner means read, write, and execute permissions.
- `r-x` for the group means read and execute permissions.
- `---` for others means no permissions.

Permissions breakdown :

- **Owner**: `anthony`
- **Group**: `anthony`

Configuration of Permissions:

- **Owner (anthony)**:
  - Can read, write, and execute (navigate into the directory).
- **Group (anthony)**:
  - Can read and execute, but cannot write.
- **Others**:
  - Cannot read, write, or execute.

Who can list files :

- **Owner**: `anthoony` has full permissions to read, which includes listing files.
- **Group**: Members of the group `anthoony` can also list files due to read permission.
- **Others**: Cannot list files as they have no permissions.

Who can create files :

- **Owner**: Only the owner, `anthoony`, can create files in the directory because write permission is exclusively granted to the owner.

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

Input and output :

```
$ ls -ld /tmp
drwxrwxrwt 20 root root 4096 mai    6 18:21 /tmp
```

- `d`: Indicates that it is a directory.
- **`rwx` for the owner**: The owner (usually `root`) has read, write, and execute permissions.
- **`rwx` for the group**: The group (usually `root` or a general group) has read, write, and execute permissions.
- **`rwt` for others**: All other users also have read, write, and execute permissions. The `t` at the end is the sticky bit.

Permissions :

- **General Write Access**: The `rwx` permissions for owner, group, and others mean that any user can create, read, and execute files in `/tmp`.
- **Sticky Bit (`t`)**: This special permission bit is crucial in a shared directory like `/tmp`. It allows all users to write files but ensures that users can only delete or rename their own files, not those of other users. This prevents unauthorized removal or alteration of files that belong to other users.

Modifying access rights

1. Create a file and initialize its permissions to `rw- --- ---` with the following commands:

```
$ touch file
$ chmod 600 file
```

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

```
rw- r-- ---
rwx r-x ---
r-- r-- r--
rwx r-- r--
rwx --- ---
```

Inputs :

Note: inputs must be placed in the same order

File creation and initialization input :

```
$ touch file
```

```
rw- r-- ---
```

```
$ chmod g+r file
```

```
rwx r-x ---
```

```
$ chmod g+x file
```

```
r-- r-- r--
```

```
$ chmod a=r file
```

```
rwx r-- r--
```

```
$ chmod u+wx file
```

```
rwx --- ---
```

```
$ chmod go-r file
```

2. Conflicting permissions - Create a file (you are going to be the owner) where the permissions are configured to

   - not allow the owner or the group to write to the file,
   - but allow the other users to write to the file.

   What does the OS do if you try to write to this file?

Creating and allocating file rights :

```
$ touch conflict_file
$ chmod 466 conflit_file
```

We can check the rights on the file (output with the command `ls -l`) :

```
$ ls -l
-r--rw-rw- 1 anthony anthony 0 mai    6 22:08 conflict_file
```

We can see that the group owner has read-only rights, while two others have additional write rights.

All that's left is to test :

```
$ echo "test write" > conflict_file
bash: conflict_file: Permission non accordée
```

If we try to access the file, the operating system displays an error message.

Giving other users access to your files

Do this exercise with another student. Both of you log in to the shared server at `ads.iict.ch` .

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

On the shared server : `ads.iict.ch` (Ubuntu 22.04.4 LTS)

Yes, my colleague is able to read the files in my home directory. To understand why, you just need to look at the permissions of the directories in `/home`:

```
/home$ ls -l
total 44
drwxr-xr-x  5 heiguser heiguser 4096 May  6 16:07 heiguser
drwxr-xr-x  6 lab0     lab0     4096 Mar 18 11:29 lab0
drwxr-xr-x 12 laba     laba     4096 May  6 15:34 laba
drwxr-xr-x  7 labb     labb     4096 Apr 14 17:17 labb
drwxr-xr-x  6 labc     labc     4096 Apr 14 10:27 labc
drwxr-xr-x  8 labd     labd     4096 Apr 12 11:27 labd
drwxr-xr-x  7 labe     labe     4096 Apr 14 22:32 labe
drwxr-xr-x  7 labf     labf     4096 Apr 29 16:57 labf
drwxr-xr-x  6 labg     labg     4096 Apr 14 22:00 labg
drwxr-xr-x  7 labh     labh     4096 May  7 19:22 labh
drwxr-xr-x  7 labi     labi     4096 Apr 14 09:08 labi
```

We can see that, based on the permissions, everyone has the right to read and execute (`r-x` at the end of the string).

This corresponds to the note in the instruction which says that since Ubuntu 21.04 home folders are denenus private but that this has been disabled on the server.

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?

To allow my colleague to read the files in my home directory, I don't need to do anything. However, to ensure that only he can read the folder in my home directory, I need to proceed as follows:

1. Make the home directory private:

   ```
   $ chmod 750 ~
   ```

2. Set a specific group for my home directory:

   ```
   $ chgrp proj_a ~
   ```

3. Add my colleague's username to the group defined for access:

   ```
   $ sudo adduser labf proj_a
   ```

   Note: Assume that only my colleague is a member of the `proj_a` group.

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?

   Note: Most Linux distributions make the users' home directories open to everyone (read access). Ubuntu adopts a new policy in release 21.04 to make home directories private, this feature has been deactivated on our server.

Translation with Instructions:

To allow my colleague to read the files in my home directory, I don't need to do anything. However, to ensure that only he can read the folder contained in my home directory, I need to proceed as follows:

1. Make the home directory private:

   ```
   $ chmod 750 ~
   ```

2. Set a specific group for my home directory:

   ```
   $ chgrp proj_a ~
   ```

3. Add my colleague's username to the group defined for access:

   ```
   $ sudo adduser labf proj_a
   ```

   Note: Assume that only my colleague is a member of the `proj_a` group.

4. Create a `shared` directory in your home directory:

   ```
   $ mkdir ~/shared
   ```

5. Set the group of the `shared` directory to `proj_a`:

   ```
   $ chgrp proj_a ~/shared
   ```

6. Assign appropriate permissions:

   ```
   $ chmod 2770 ~/shared
   ```

   - `2`: Activates the "setgid" bit so that all files created inherit the group `proj_a`.
   - `7` (owner): `rwx` (read, write, execute)
   - `7` (group): `rwx` (read, write, execute)
   - `0` (others): `---` (no permissions)

Summary of Commands:

```
$ chmod 750 ~
$ chgrp proj_a ~
$ sudo adduser labf proj_a
$ mkdir ~/shared
$ chgrp proj_a ~/shared
$ chmod 2770 ~/shared
```

Find

`find` is a powerful Unix command to search the files on a computer by name, by size, or other criteria.

1. Pro tip: If you run `find` without any search criteria, it will simply display everything it comes across in its recursive traversal. So to list recursively everything in the current directory, type simply

   ```
   find .
   ```

   What does find do with hidden files or directories?

By default, the command `find` finds and displays all files and directories, including hidden files and folders.

In Unix and Linux systems, files or directories are considered hidden if their names start with a "`.`". These files are generally used to store user configuration or system information that should not be accidentally modified or deleted.

When you run `find .`, the command lists all entries in the current directory and all its subdirectories, including those whose names start with a dt. For example, this would include files like `.bashrc`, `.git`, and directories like `.config`.

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

- that end in `.c` , `.cpp` or in `.sh`
- that are executable
- that have not been modified since more than two years
- that have not been accessed since more than two years
- that have not been accessed since more than three years and that are bigger than 3 MB (good candidates for cleanup)

Display all the directories in your home directory

- that are called `.git` (probably the root of a git repository)

Files ending in `.c`, `.cpp`, or `.sh`:

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

Executable Files:

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

Files not modified for more than two years:

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

Note : We assume that a year is always 365 days long.

Files not accessed for more than three years and larger than 3 MB:

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

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:

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

The first command is generally more correct and reliable:

- Safety and Reliability: The first command does not rely on shell globbing (expansion of `*`), which can be problematic with a large number of files or special filenames (e.g., filenames starting with a dash).
- Consistency: The first command consistently applies to any case, regardless of the presence or specifics of the files in the directory.
- Direct Control: It directly searches within the specified directory (.) and is not influenced by any peculiarities of filename patterns that could interfere with command line parsing.

Therefore, while both commands could potentially work in specific scenarios, the first command is more universally robust and should be preferred for general use to find files containing a specified text. It avoids issues with shell expansions and ensures that all files are correctly processed without assuming anything about their names or count.

Task 2: Display world-writable files

Context

While working with their files users may accidentally set permissions that are too loose and create security problems. As a system administrator you want to provide your users with a tool that they can use to scan their files, detect problematic permissions and correct them.

Tool for security checks

The tool will allow users to detect files or directories that are "world-writable". World-writable means the write bit is set for "others". This is obviously a security risk. Create a test directory that you will use while developing the tool. Create a directory named `test_dir` and in it create a few files and subdirectories. Change the permissions on some files and some directories using `chmod` so that the write bit is set for "others". Use the `find` command on the test directory to list all files and directories. Then modify the command to find only the world-writable ones. Write a script named `fix_permissions` that for the time being only displays the world-writable files and directories. The output should look as follows:

```
The following files/directories are world-writable:
test_dir/foo
test_dir/dir1
test_dir/dir2
```

Setting up the directories, files and permissions

```
$ mkdir test_dir && cd test_dir
$ touch file1.txt file2.txt file3.txt
$ mkdir dir1 dir2
$ chmod o+w file1.txt dir1
```

Using find to list all world writable files and directories

Files:

```
$ find . -type f -perm -o=w -print
```

output:

```
./file1.txt
```

Directories:

```
$ find . -type d -perm -o=w -print
```

output:

```
./dir1
```

Script:

```
#!/bin/bash

echo "The following files/directories are world-writable:"

# Find and display world-writable files
find ./test_dir -type f -perm -o=w -print

# Find and display world-writable directories
find ./test_dir -type d -perm -o=w -print
```

output:

```
The following files/directories are world-writable:
./test_dir/file1.txt
./test_dir/dir1
```

Task 3: Pass the directory as an argument

The user should be able to specify which directory the tool should analyze. 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.

Script

```bash
#!/bin/bash

# Check if exactly one argument is passed
if [ $# -ne 1 ]; then
    echo "Usage: $0 <directory>"
    exit 1
fi
# Check if the argument is a valid directory
if [ ! -d "$1" ]; then
    echo "Error: '$1' is not a valid directory."
    exit 1
fi
writable_files=$(find "$1" -type f -perm -o=w)
writable_dirs=$(find "$1" -type d -perm -o=w)
# Check if the list is empty
if [ -z "$writable_files" ] && [ -z "$writable_dirs" ]; then
    echo "No world-writable files or directories found."
    exit 0
else
    echo "The following files/directories are world-writable:"
    echo "$writable_files"
    echo "$writable_dirs"
fi
```

Tests

Testing with no arguments:

`./fix_permissions`

output:

`Usage: ./fix_permissions <directory>`

Testing with invalid directory:

`./fix_permissions non_existent_directory`

output:

`Error: 'non_existent_directory' is not a valid directory.`

Testing with `test_dir`

```bash
# Valid directory
./fix_permissions /path/to/your/test_dir
```

output:

```
The following files/directories are world-writable:
test_dir/file1.txt
test_dir/dir1
```

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. Modify the script such that after displaying the world-writable files it asks the user the question

`Do you want the permissions to be fixed (y/n)?`

If the user responds affirmatively the script proceeds and removes the permissions.

Note: The tool should ask only once for all the files, not for each file individually.

Tip: To avoid having to re-create the initial state of the test directory every time you run the script do the following: Run the script on a copy of the test directory that you throw away after use.

Script

```bash
#!/bin/bash

# Check if exactly one argument is passed
if [ $# -ne 1 ]; then
    echo "Usage: $0 <directory>"
    exit 1
fi
# Check if the argument is a valid directory
if [ ! -d "$1" ]; then
    echo "Error: '$1' is not a valid directory."
    exit 1
fi

writable_files=$(find "$1" -type f -perm -o=w)
writable_dirs=$(find "$1" -type d -perm -o=w)

# Check if the list is empty
if [ -z "$writable_files" ] && [ -z "$writable_dirs" ]; then
    echo "No world-writable files or directories found."
    exit 0
else
    echo "The following files/directories are world-writable:"
    echo "$writable_files"
    echo "$writable_dirs"
    # Ask the user if they want to fix the permissions
    echo "Do you want the permissions to be fixed (y/n)?"
    read answer

    if [ "$answer" = "y" ]; then
        # If the user agrees, remove world-writable permissions from files
        for file in $writable_files; do
            chmod o-w "$file"
        done
        # Remove world-writable permissions from directories
        for dir in $writable_dirs; do
            chmod o-w "$dir"
        done
        echo "Permissions have been fixed."
    else
        echo "No changes made."
    fi
fi
```

Tests

```
# Copy the test dir
cp -r test_dir test_dir_backup

# Sep world writable permissions to file1, dir1 and display permissions
cd test_dir_backup && chmod o+w file1.txt dir1 && ls -l && cd ..
```

output:

```
total 8
drwxrwxrwx 2 tim tim 4096 Mai  1 10:59 dir1
drwxrwxr-x 2 tim tim 4096 Mai  1 10:59 dir2
-rw-rw-rw- 1 tim tim    0 Mai  1 10:59 file1.txt
-rw-rw-r-- 1 tim tim    0 Mai  1 10:59 file2.txt
-rw-rw-r-- 1 tim tim    0 Mai  1 10:59 file3.txt
```

Launching the script:

```
./fix_permissions test_dir_backup/
```

Output:

```
The following files/directories are world-writable:
test_dir_backup/file1.txt
test_dir_backup/dir1
Do you want the permissions to be fixed (y/n)?
```

Output by selecting `y`:

```
Permissions have been fixed.
```

Check the new permissions:

```
cd test_dir_backup && ls -l && cd ..
```

Output:

```
total 8
drwxrwxr-x 2 tim tim 4096 Mai  1 10:59 dir1
drwxrwxr-x 2 tim tim 4096 Mai  1 10:59 dir2
-rw-rw-r-- 1 tim tim    0 Mai  1 10:59 file1.txt
-rw-rw-r-- 1 tim tim    0 Mai  1 10:59 file2.txt
-rw-rw-r-- 1 tim tim    0 Mai  1 10:59 file3.txt
```

We can see that `dir1` and `file1` have their permission fixed.

***Bonus:* option to automatically fix permissions**

In this "Bonus", we've added the option to specify the `-y` option in the script to pre-select the user prompt and automatically fix the permissions.

```
#!/bin/bash

# Initialize default answer as empty
answer=""

# Check for the '-y' option
if [ "$1" = "-y" ]; then
    answer="y"
    shift  # Shift the arguments to the left
fi
# Check if exactly one argument is passed
if [ $# -ne 1 ]; then
    echo "Usage: $0 [-y] <directory>"
    exit 1
fi
# Check if the argument is a valid directory
if [ ! -d "$1" ]; then
    echo "Error: '$1' is not a valid directory."
    exit 1
fi

writable_files=$(find "$1" -type f -perm -o=w)
writable_dirs=$(find "$1" -type d -perm -o=w)

# Check if the list is empty
if [ -z "$writable_files" ] && [ -z "$writable_dirs" ]; then
    echo "No world-writable files or directories found."
    exit 0
else
    echo "The following files/directories are world-writable:"
    echo "$writable_files"
    echo "$writable_dirs"

    if [ -z "$answer" ]; then
        # Ask the user if they want to fix the permissions
        echo "Do you want the permissions to be fixed (y/n)?"
        read answer
    fi

    if [ "$answer" = "y" ]; then
        # If the user agrees, remove world-writable permissions from files
        for file in $writable_files; do
            chmod o-w "$file"
        done
        # Remove world-writable permissions from directories
        for dir in $writable_dirs; do
            chmod o-w "$dir"
        done
        echo "Permissions have been fixed."
    else
        echo "No changes made."
    fi
fi
```

Test

```
./fix_permissions -y test_dir
```

output:

```
The following files/directories are world-writable:
test_dir/file1.txt
test_dir/dir1
Permissions have been fixed.
```

Task 5: Display group-writable files

To prepare for this task, create two groups on your local machine by becoming the superuser:

```
sudo addgroup proj_a
sudo addgroup proj_b
```

Then add yourself to both groups (replace albert.einstein with your user name):

```
sudo adduser albert.einstein proj_a
sudo adduser albert.einstein proj_b
```

In most companies the employees work in teams and they need to share documents, source code or other files with their colleagues. To enable this sharing on the Unix file system level a user needs to change the permissions on files and directories.

A file or directory that is shared in a team

- is assigned a group that corresponds to the team (each team member becomes member of the group),
- has write permissions for the group.

Sometimes a user forgets that he shares certain files with others. It is helpful to remind the user which files and directories are shared with the teams. Extend the script to show all files and directories that are writable for the group. However do not include files or directories assigned to the personal group of the user (the personal group is the one that has the same name as the user name).

The script should produce output like the following:

```
The following files/directories are writable for groups:
test_dir/dir3
test_dir/baz
```

To test the script extend your test directory with files that belong to project groups and are group-writable.

Hints:

- On Ubuntu Linux the personal group of the user has the same name as the user (this is a standard practice on almost all modern Linux distributions).
- The name of the current user is available in the environment variable $USER .
- You do not need to verify which groups the user is a member of. When the file has a group owner that is not the personal group of the user there is a potential security problem, and the file should be displayed.

The following script has been added to fix_permissions:

```
# Task 5: Display group-writable files
# Find all group-writable files and directories
writable_items=$(find "$1" \( -type f -o -type d \) -perm -g+w)

# Exclude files and directories assigned to the personal group of the user
personal_group=$(id -gn)
filtered_items=""
while IFS= read -r item; do
    group=$(stat -c '%G' "$item")
    if [ "$group" != "$personal_group" ]; then
        filtered_items+="\n$item"
    fi
done <<< "$writable_items"

# Check if the list is empty after filtering
if [ -z "$filtered_items" ]; then
    echo "No group-writable files or directories found."
else
    echo "The following files/directories are writable for groups:"
    echo -e "$filtered_items"
fi
```