Intro to Cybersecurity - Lab 4
Access Control: UNIX Permissions and sudo Security

Aymane Sghier, Jun Wang, Zhipeng Yang

October 8, 2025

1 Introduction

This lab explores UNIX file permissions and access control mechanisms. Our team investigated proper filesystem permission configurations, analyzed sudo security vulnerabilities, and examined privilege escalation attacks using the FrobozzCo permissions test system on SPHERE.

2 Task 4.2: UNIX File Permissions and sudo

For Section 4.1 and 4.2, I made 4 different cases for file permission.

Case 1: Permission of files.I made two files and set different permission for them.

Case 2: Permission of groups: I use chown and chmod command to create a shared group,and menbers of the group can read/write/execute, and others have no access.

Case 4:wideopen permission: this is dangerous, and everyone can read-/write/execute the files.

Case 4: Special permissions: Setuid:user run as root. Setgid:run with the permission of the file's group. sticky:protect files from deletion by others

3 Task 4.3: Home Directory Security

In this section, we set different permissions for users and group members. Commands using:
mkdir for creat directory.

groupadd for creat groups.

adduser –home for adding users.

When I add user to /emp,I do not know why it add users into /home.Therefore,I us ecommand mv ,move them to /emp, and then add the users into different groups.

usermod -aG for adding users into group.

chmod used for setting the permission.

chown used for changing the owner of the groups or files.

The command lines and the answers that I have uploaded in the zip file. permissions-answers.txt for answers, and tarball.png for tarball results screenshot.

4 Task 4.4: The Ballot Box

4.1 Implementation

Create a directory /ballots where

- All users can submit (write) ballots
- But cannot see or read anyone else files
- And even privileged users(like users in wheel) have no direct access

4.2 Short Answer 1

Question: Is there any way that employees can read the ballots of other users? If so, what could be done to stop this? If not, what prevents them from reading them?

Answer: No, employees cannot read other users' ballots. They lack read permission on the directory, so they cannot list its contents or open existing files they don't own. The 'x' (execute) bit allows entering the directory, but without 'r' they can't see filenames.

4.3 Short Answer 2

Question: What does the 'x' bit mean when applied to directories?

Answer: For directories, the 'x' bit means "search permission" — it allows users to access files or subdirectories within if they know the exact name. Without 'x', even knowing the filename, they cannot cd into or open it.

4.4 Analysis

- Everyone can write their ballot
- No one can read or list other's ballots
- wheel group has no access at all

5 Task 4.5: The TPS Reports Directory

5.1 Methodology

5.2 Implementation

Create a shared working directory /tpsreports where

- Owned by user tps, group wheel
- Only members of wheel(and tps) have access
- Files created inherit group ownership
- No one can delete files they don't own

5.3 Short Answer 3

Question: Which users on the system can delete arbitrary files in the /tpsreports directory?

Answer: Only the directory owner (tps) and root can delete arbitrary files. Group members can delete only files they own, due to the sticky bit.

5.4 Short Answer 4

Question: Is there any way that non-wheel employees can read files in the /tpsreports directory?

Answer: No, non-wheel users cannot read any files inside. They lack both read (r) and execute (x) permissions on the directory, preventing traversal or listing.

5.5 Short Answer 5

Question: What do '0' permissions mean for the owner of a directory? What privileges do they have over files in that directory?

Answer:A permission of '0' means the owner has no rights at all—no read, write, or execute. The owner cannot list, access, or delete anything inside the directory unless privileges are granted elsewhere (e.g., via root).

5.6 Analysis

- Directory shows drwxrws–t tps wheel ... /tpsreports
- Both tps and wheel members can read/write
- Files stay in the wheel group
- Non-members have no access
- Within wheel, users cannot delete other's work

6 Task 4.6: sudo - Editing Configuration Files

6.1 Short Answer 6

Question: Is this safe? Why or why not? If it is not safe, is there a better way to give larry this access? If it is safe, how do you know it is safe?

Answer: No, this is not safe. Although it appears to limit `larry` to editing a single file, the command `vim` allows opening a shell (for example, with `:!bash`), giving `larry` full root access. The `NOPASSWD` option further removes any authentication, making privilege escalation easier.

Safer Alternative

Instead of allowing `vim`, use a non-interactive command that cannot open a shell:

```
larry ALL=(ALL) NOPASSWD: /usr/bin/tee /etc/httpd/httpd.conf
```

Then `larry` can safely update the file using:

```
cat new_httpd.conf | sudo tee /etc/httpd/httpd.conf
```

7 Task 4.7: sudo - Restarting System Processes

7.1 Short Answer 7

Question: Assuming the init script /etc/init.d/httpd has no unknown vulnerabilities, is it safe to grant larry sudo access to the command /etc/init.d/httpd restart? If this is not secure, explain why.

Answer:

No, it is NOT safe to grant larry sudo access to /etc/init.d/httpd restart. The sudoers entry `larry ALL=(root) NOPASSWD: /etc/init.d/httpd restart` doesn't properly restrict which arguments can be passed to the script. Without explicit argument matching, larry could potentially execute other commands like `/etc/init.d/httpd stop` (halting the web server) or `/etc/init.d/httpd` `start` instead of just restart. Init scripts are shell scripts that execute with full root privileges via sudo, making them vulnerable to exploitation through environment variables such as PATH, LD_PRELOAD, and IFS, or through sourcing external configuration files that may have inadequate permission controls. Even assuming the script has no known vulnerabilities, shell scripts running as root present a fundamentally larger attack surface than compiled binaries because the shell interpreter itself adds complexity where exploitation opportunities may exist. A better and more secure alternative would be to use systemctl: `larry ALL=(root) NOPASSWD: /usr/bin/systemctl restart httpd`, as systemctl is a compiled binary with significantly reduced attack surface compared to shell scripts.

8 Task 4.8: UNIX and sudo - Two Wrongs Make a Much Bigger Wrong

8.1 Short Answer 8

Question: Is there some way that moe or curly could subvert this system to gain root privileges? If not, how do you know this is true?

Answer:

Yes, both moe and curly can gain root privileges by exploiting the shell initialization process during admin user logins. According to the lab configuration in section 4.3, moe is in bwk's group and can write to /admins/bwk/, while curly is in dmr's group and can write to /admins/dmr/. Since bwk and dmr are both members of the wheel group, they have full sudo access with NOPASSWD enabled. The attack works as follows: moe can add malicious code to bwk's .bashrc file by executing: `echo 'cp /bin/bash /tmp/moe_shell' >> /admins/bwk/.bashrc` followed by `echo 'chmod 4755 /tmp/moe_shell' >> /admins/bwk/.bashrc`. When bwk logs in next, the .bashrc file executes automatically with bwk's privileges, creating a copy of bash with the setuid bit set (the 4755 permission sets the setuid bit plus rwxr-xr-x permissions). After bwk logs in, moe can execute `/tmp/moe_shell -p` (the -p flag tells bash to preserve the effective user ID) to obtain a shell running with bwk's privileges. Since bwk is in the wheel group with sudo access, moe can then execute `sudo su -` to escalate to root without needing a password. The same attack vector works for curly exploiting dmr's account. This vulnerability exists because shell initialization files (.bashrc, .bash_profile, .profile) execute during the login process—specifically between when the user enters the correct password and when they receive an interactive shell—and group write permissions allow non-privileged users to modify these critical files belonging to privileged users.

9 Conclusion

This lab provided hands-on experience with UNIX access control mechanisms and demonstrated critical security principles. Through investigating sudo configurations and permission structures, we identified how seemingly reasonable administrative decisions can create serious privilege escalation vulnerabilities. The principle of least privilege and proper separation of duties are essential to prevent the types of vulnerabilities explored in this lab.

10 Appendix: Commands and Screenshots

All commands executed and their outputs have been included as separate files in the submission.