CSPP EXP 8

8.1 Aim: Implementing File, Process, and Application Permissions: Apply the principle of least privilege using standard operating system security controls.

8.2 Course Outcome: Apply least privilege principles to files, processes, and applications in order to minimize attack surface, enforce access control, and secure system operations.

8.3 Lab Objective: To configure, modify, and verify permissions for files, processes, and applications so that only authorized users and programs can access or execute them securely.

8.4 Requirements:

- Operating System: Linux (Ubuntu/Kali), Windows, or macOS
- Tools/Commands:
  - Files: ls, chmod, chown, icacls (Windows)
  - Processes: ps, top, kill, Task Manager (Windows)
  - Applications: Linux AppArmor / SELinux, Windows UAC, Run as Administrator
- User accounts: At least 2 (Admin + Standard user)
- Test files, processes, and applications

8.5 Theory:

The Principle of Least Privilege (PoLP) ensures that users, processes, and applications only have the minimum level of access necessary to perform their tasks.

- File Permissions: Define who can read, write, or execute files.
  - Linux: Owner, Group, Others (chmod, chown).
  - Windows: ACLs (Read, Write, Modify, Full Control).

- Process Permissions: Control which users can start, monitor, or terminate processes.
  - Linux: Each process has a UID/GID; normal users cannot kill root-owned processes.
  - Windows: Processes run with the privileges of the logged-in user; some require elevated/admin rights.

● Application Permissions: Limit application capabilities using security frameworks.
  ○ Linux: SELinux/AppArmor policies restrict app access to files and resources.
  ○ Windows: User Account Control (UAC) prevents applications from running with admin rights unless approved.

By configuring permissions at all three levels, organizations can prevent privilege escalation, malware execution, and unauthorized data access.

8.6 Tasks:

Part A – File Permissions
1. Create test users (user1, user2).
2. Create a secure file (e.g., /home/user1/confidential.txt).
3. Use chmod 600 so only user1 can read/write.
4. Switch to user2 and attempt access → should be denied.

Part B – Process Permissions
5. Run a process (e.g., ping google.com) under user1.
6. Switch to user2 and try to terminate user1's process → should be denied.
7. Use sudo (Linux) or Task Manager (Windows) to demonstrate how only privileged users can stop certain processes.

Part C – Application Permissions
8. Install or select a test application.
9. On Linux: Apply AppArmor/SELinux policy to restrict file access.
10. On Windows: Run application as Standard User and observe restrictions, then run as Administrator with UAC approval.
11. Demonstrate how least privilege prevents unauthorized modifications or system-level changes.

8.7 Output :

Part A :

```
┌──(kali㉿kali)-[~]
└─$ sudo adduser user1
[sudo] password for kali:
info: Adding user `user1' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `user1' (1001) ...
info: Adding new user `user1' (1001) with group `user1 (1001)' ...
info: Creating home directory `/home/user1' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for user1
Enter the new value, or press ENTER for the default
        Full Name []: Swayam
        Room Number []: 1
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
info: Adding new user `user1' to supplemental / extra groups `users' ...
info: Adding user `user1' to group `users' ...

┌──(kali㉿kali)-[~]
└─$ sudo adduser user2
info: Adding user `user2' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `user2' (1002) ...
info: Adding new user `user2' (1002) with group `user2 (1002)' ...
info: Creating home directory `/home/user2' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for user2
Enter the new value, or press ENTER for the default
        Full Name []: Swayam 2
        Room Number []: 2
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
info: Adding new user `user2' to supplemental / extra groups `users' ...
info: Adding user `user2' to group `users' ...
```

```
┌──(kali㉿kali)-[~]
└─$ sudo -i -u user1 bash -c 'mkdir -p /home/user1; echo "TOP SECRET - DO NOT SHARE" > /home/user1/confidential.txt'

┌──(kali㉿kali)-[~]
└─$ sudo chown user1:user1 /home/user1/confidential.txt

┌──(kali㉿kali)-[~]
└─$ sudo chmod 600 /home/user1/confidential.txt
```

```
┌──(kali㉿kali)-[~]
└─$ sudo -i -u user1 ls -l /home/user1/confidential.txt

-rw------- 1 user1 user1 26 Oct  2 02:42 /home/user1/confidential.txt

┌──(kali㉿kali)-[~]
└─$ sudo -i -u user1 cat /home/user1/confidential.txt
# Expected: "TOP SECRET - DO NOT SHARE"

TOP SECRET - DO NOT SHARE

┌──(kali㉿kali)-[~]
└─$ sudo -i -u user2 cat /home/user1/confidential.txt
# Expected: cat: /home/user1/confidential.txt: Permission denied

cat: /home/user1/confidential.txt: Permission denied
```

Part B :

```
┌──(kali㉿kali)-[~]
└─$ sudo -i -u user1 bash -c 'ping -c 200 google.com > /tmp/ping_user1.log 2>&1 & echo $!'
# This prints the PID; note it (e.g., 12345)

7126

┌──(kali㉿kali)-[~]
└─$ ps -u user1 -o pid,cmd | grep ping
   7126 ping -c 200 google.com

┌──(kali㉿kali)-[~]
└─$ sudo -i -u user2 bash -c 'kill 7126'
bash: line 1: kill: (7126) - Operation not permitted

┌──(kali㉿kali)-[~]
└─$ sudo kill 7126

┌──(kali㉿kali)-[~]
└─$ ps -p 7126 || echo "process not found"
    PID TTY          TIME CMD
process not found
```

Part C :

```
┌──(kali㉿kali)-[~]
└─$ sudo aa-status
apparmor module is loaded.

┌──(kali㉿kali)-[~]
└─$ sudo tee /usr/local/bin/testapp.sh >/dev/null <<'EOF'
#!/bin/bash
echo "[testapp] running as: $(id -un)"
echo "[testapp] trying to read confidential file ... "
if cat /home/user1/confidential.txt 2>/tmp/testapp_err; then
  echo "[testapp] read succeeded"
else
  echo "[testapp] read failed"
  echo "cat error:"
  sed -n '1,50p' /tmp/testapp_err
fi
sleep 1
EOF

sudo chmod +x /usr/local/bin/testapp.sh

┌──(kali㉿kali)-[~]
└─$ sudo -i -u user1 /usr/local/bin/testapp.sh
# Expected: read succeeds (because file owner is user1 and perms are 600)
[testapp] running as: user1
[testapp] trying to read confidential file ...
TOP SECRET - DO NOT SHARE
[testapp] read succeeded
```

```
┌──(kali㉿kali)-[~]
└─$ sudo aa-genprof /usr/local/bin/testapp.sh
Updating AppArmor profiles in /etc/apparmor.d.
Writing updated profile for /usr/local/bin/testapp.sh.
Setting /usr/local/bin/testapp.sh to complain mode.

Before you begin, you may wish to check if a
profile already exists for the application you
wish to confine. See the following wiki page for
more information:
https://gitlab.com/apparmor/apparmor/wikis/Profiles

Profiling: /usr/local/bin/testapp.sh

Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" option below in
order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the
opportunity to choose whether the access should be
allowed or denied.

[(S)can system log for AppArmor events] / (F)inish
Reading log entries from /var/log/syslog.

Profile:  /usr/local/bin/testapp.sh
Execute:  /usr/bin/id
Severity: unknown
```

```
Finished generating profile for /usr/local/bin/testapp.sh.

┌──(kali㉿kali)-[~]
└─$ sudo aa-enforce /usr/local/bin/testapp.sh
Setting /usr/local/bin/testapp.sh to enforce mode.
```

8.8 Conclusion :

In this experiment, we successfully applied the **principle of least privilege (PoLP)** to files, processes, and applications in Kali Linux. By configuring strict file permissions, we ensured that sensitive data could only be accessed by authorized users. Through process-level controls, we demonstrated that standard users cannot interfere with or terminate processes owned by others, preventing unauthorized system manipulation. Finally, by implementing AppArmor profiles, we restricted an application's access to critical files, enforcing security beyond basic UNIX permissions. Overall, the lab highlighted how properly managing permissions at multiple levels minimizes the attack surface, enforces access control, and strengthens system security against potential threats