## User & Permissions Management - Processes & Jobs - Processes

**whoami** – Display current user.

**id** – identifies your login name and group and any supplementary group you belong to.

**sudo** – short for “superuser do”, execute commands with root privileges.

**passwd** – Change user password. It is important to regularly change passwords as a fundamental practice in maintaining secure systems.

**chmod**– Modify file permissions of a file or directory .

This is crucial for securing files and directories by controlling who can read, write, or execute them. For example, **chmod 700 filename** would give the owner read, write, and execute permissions, while all other users would have no permissions

chown – Change file ownership of files and directories.

To use **chown**, you must be the owner of the file. Only root can use **chown** to change ownership. Ordinary users cannot give their files to someone else. Using **chgrp**, ordinary users can change the group ownership of a file if they own the file and they are a member of the target group.

This is another important tool for securing file and directory access. For example, **chown user:group filename** would change the ownership of the file to the specified user and group.

usermod

The **usermod** command allows an administrator to modify a user account. This can include adding the user to a new group (which may have certain permissions), changing the user’s home directory, or even changing the username.

find

While not specifically a security command, f**ind** is a powerful tool for locating files and directories. It can be used in conjunction with other commands to search for files with specific permissions or ownership.

last

The **last** command shows the last logins on the system. This can sometimes help to detect any unusual login activity.

history

The **history**command displays the command history of the current user. It’s beneficial for auditing purposes, allowing you to see what commands have been run previously.

**Process and System Monitoring**

* ps aux – List running processes.
* top – Display real-time system processes.
* kill – Terminate a process.
* df -h – Show disk space usage.
* uptime – Display system uptime.

**Networking Commands**

* ifconfig / ip a – View and configure network interfaces.
* netstat -tulnp – Display active connections and listening ports (ss also used for same purpose).
* nmap – Scan networks for open ports.
* ping – Check network connectivity.
* tcpdump – Capture and analyse network traffic.

## Permissions & Users

### Commands: chmod, chown, umask, whoami, id, groups

Exercise: Create a file and make it readable only by the owner; check with ls -l.

A screen shot of a computer

AI-generated content may be incorrect.

Run **whoami**, **id**, and **groups** to check your current user and group.

### Question: How do permissions prevent data leaks?

Linux file permissions control **who can read, write, or execute** files. Permissions act as a first layer of defence, stopping unauthorised access to sensitive logs, credentials, or configuration files.

## Processes & Jobs

### Commands: ps, top, kill, jobs, fg, bg

Start a program that runs for a while. In this example, the program will run for 100 seconds. The job is run in the background.

A black rectangle with white text

AI-generated content may be incorrect.

Use jobs to see the background job.



Use **ps aux | grep sleep** to find the **PID** of the process.

A black and white text

AI-generated content may be incorrect.

Kill the process:

A black background with white text

AI-generated content may be incorrect.

**Question**: Can you think of a reason why the output shows there is no process to kill?

### Question: How might malware hide as a process?

Malware often runs as a background process, disguising itself to avoid detection. Detecting malware often means auditing running processes for anomalies (strange names, unusual paths, odd parent/child relationships)

* It may use a legitimate-sounding name (ssh, cron, kworker) so it blends into normal system activity.
* It may run with low CPU usage so it doesn’t stand out in top or ps.
* Advanced malware may even unlink itself from process listings, though most beginner-level malware just relies on looking “normal.”

## Useful Utilities

### Tools: tar, gzip, strings, file, hexdump

Create a new folder project with sample files.



Compress the folder into project.tar.gz.



Extract it into a new directory unpack.



Create a mock binary file for analysis.



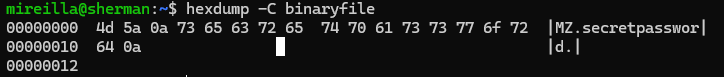
Use **file binaryfile** to identify what kind of file it is.



Run **strings binaryfile** to extract printable text (e.g., "secretpassword").



Run hexdump -C binaryfile to see the raw hexadecimal contents.



### Question: How could **strings** and **hexdump** help detect malware?

These tools reveal what is hidden inside binary files. Using both together helps analysts **reverse-engineer malware samples**, spotting hidden instructions or data that normal text editors can’t show.

**Strings** extracts **printable text** (like hardcoded IPs, URLs, or passwords).

* Example: A suspicious binary might contain http://malicious.example.com or secret\_key=....

**hexdump** shows the **raw binary data** in hex + ASCII side by side.

* Useful to see magic numbers (file signatures) or non-printable payloads.