Javan Pahlavan Mentorship

Course Assignment – T(6)

Prepared By:

Mahdi Siamaki

(siamaki.me@gmail.com)

Instructor:

Mr. Mahdi Bahrami

July 2025

Question 1:

What is the purpose of the /etc/passwd file, and how is it related to /etc/shadow?

These two files are fundamental to user account management on a Linux system. They work together to store user information, but they separate public data from sensitive data for security reasons.

/etc/passwd

- **Purpose:** This is the primary user database. It contains a list of all user accounts on the system, along with their basic attributes. This file must be readable by all users so that the system can map User IDs (UIDs) to human-readable usernames (e.g., for ls -l command output).
- **File Format:** Each line represents one user and contains seven colon-separated fields:
    1. **Username:** The name used to log in (e.g., alice).
    2. **Password:** A placeholder, almost always an x. The x indicates that the actual encrypted password is stored in /etc/shadow.
    3. **User ID (UID):** A unique number for the user. UID 0 is always for the root user.

4. **Group ID (GID):** The primary group ID for the user.
5. **GECOS/Comment:** User's full name, office number, etc. This field is optional.
6. **Home Directory:** The absolute path to the user's home directory (e.g., /home/alice).
7. **Login Shell:** The path to the user's default shell (e.g., /bin/bash).

- **Example Line:** alice:x:1001:1001:Alice Smith:/home/alice:/bin/bash

/etc/shadow

- **Purpose:** This file securely stores the encrypted (hashed) passwords for user accounts, along with password aging information.
- **Security:** Crucially, this file is only readable by the root user. This separation prevents unprivileged users from accessing the password hashes, which could otherwise be subjected to offline cracking attempts.
- **File Format:** Each line corresponds to a user in /etc/passwd and contains nine colon-separated fields:
    1. **Username:** The user's login name (this links it to /etc/passwd).
    2. **Encrypted Password:** The hashed password string. If it's ! or *, the account is locked.

3. **Last Password Change:** Days since the epoch (Jan 1, 1970) that the password was last changed.
4. **Minimum Password Age:** Minimum number of days required between password changes.
5. **Maximum Password Age:** Maximum number of days the password is valid.
6. **Warning Period:** Number of days before password expiration that the user is warned.
7. **Inactivity Period:** Number of days after password expiration that the account is disabled.
8. **Expiration Date:** Days since the epoch when the account will be disabled.
9. **Reserved Field:** Unused.

- **Example**

  Line: alice:6somerandomsalt$longencryptedstring:19253:0:99999:7:::

Relationship: The two files are linked by the **username**. When a user tries to log in, the system reads /etc/passwd to get basic info. It sees the x in the password field, which tells it to look up the corresponding username in /etc/shadow to verify the provided password against the stored hash.

Question 2:

How can I enforce a password change policy for users every 100 days?

You can enforce this policy using the chage (change age) command, which modifies user password expiry information.

For an Existing User:

To force a specific user named bob to change their password at least every 100 days, you would use the -M (maximum days) flag:

```
sudo chage -M 100 bob
```

You can verify the change by running sudo chage -l bob, which will display all password aging settings for that user.

```
$ sudo chage -l bob
Last password change                    : Sep 21, 2023
Password expires                        : Dec 30, 2023  <-- 100 days from last change
Password inactive                       : never
Account expires                         : never
Minimum number of days between password change    : 0
Maximum number of days between password change    : 100
Number of days of warning before password expires  : 7
```

For All New Users (System-wide Default):

To make this the default policy for any new user created on the system, you need to edit the /etc/login.defs file. Find the PASS_MAX_DAYS variable and set its value to 100.

```
# Edit /etc/login.defs

...

# Maximum number of days a password may be used.

PASS_MAX_DAYS  100

...
```

Any user created with useradd *after* this change will automatically inherit this 100-day password expiration policy.

Question 3:

Create a cron job that collects all system log files (/var/log/syslog) every day at 3:00 AM and archives them weekly on Friday night. Then move the weekly archive to another location.

This task requires two separate cron jobs and a small script for the weekly task.

Step 1: Create Directories for Archives

It's good practice to have dedicated directories for the daily and weekly archives.

```
sudo mkdir -p /var/archives/daily

sudo mkdir -p /var/archives/weekly
```

Step 2: Create the Daily Cron Job

This job will create a compressed archive of the current syslog and any rotated ones (syslog*) every day at 3:00 AM.

Edit the crontab for the root user: sudo crontab -e. Add the following line:

```
# Archive syslog files daily at 3:00 AM

0 3 * * * /usr/bin/tar -czf /var/archives/daily/syslog-$(date +\%Y-\%m-\%d).tar.gz /var/log/syslog*
```

- 0 3 * * *: Cron schedule for "at 0 minutes past the 3rd hour, every day".
- $(date +\%Y-\%m-\%d): Creates a date-stamped filename like syslog-2023-09-21.tar.gz. The % must be escaped with a \ in a crontab.

Step 3: Create a Script for the Weekly Task

This script will bundle all the daily archives into a single weekly archive, move it, and clean up the daily files.

Create a script file /usr/local/bin/archive_weekly.sh:

```
#!/bin/bash



# Define directories

DAILY_ARCHIVE_DIR="/var/archives/daily"

WEEKLY_ARCHIVE_DIR="/var/archives/weekly"

REMOTE_STORAGE="/mnt/backup_server/logs" # The other location



# Create a filename for the weekly archive (e.g., weekly-2023-W38.tar.gz)

WEEKLY_FILENAME="weekly-$(date +%Y-W%V).tar.gz"

WEEKLY_ARCHIVE_PATH="${WEEKLY_ARCHIVE_DIR}/${WEEKLY_FILENAME}"



# Find daily archives and bundle them into the weekly archive

# The `|| exit 1` will stop the script if the tar command fails

/usr/bin/tar -czf "${WEEKLY_ARCHIVE_PATH}" -C "${DAILY_ARCHIVE_DIR}" . || exit 1



# Move the new weekly archive to the remote location

/usr/bin/mv "${WEEKLY_ARCHIVE_PATH}" "${REMOTE_STORAGE}/"



# If the move was successful, clean up the old daily archives

if [ $? -eq 0 ]; then

  /usr/bin/rm -f ${DAILY_ARCHIVE_DIR}/*.tar.gz

fi
```

Make the script executable: sudo chmod +x /usr/local/bin/archive_weekly.sh.

Step 4: Create the Weekly Cron Job

This job runs the script on Friday night at 11:00 PM. Edit the root crontab again (sudo crontab -e) and add:

```
# Run the weekly log archival script on Fridays at 11 PM

0 23 * * 5 /usr/local/bin/archive_weekly.sh
```

- 0 23 * * 5: Cron schedule for "at 0 minutes past the 23rd hour, on the 5th day of the week (Friday)".

Question 4:

Recreate question 3 using systemd-run instead of a traditional cron job.

The request specifies systemd-run, which is typically for running one-off or transient commands. It can, however, create transient timers with the --on-calendar flag. A more robust and standard systemd solution uses persistent timer and service units. Both methods are shown below.

Method 1: Using systemd-run (Transient Timers)

This method directly uses systemd-run as requested. The drawback is that these scheduled tasks are transient and will **not persist after a reboot**.

- **Daily Job (3:00 AM):**

```
sudo systemd-run --on-calendar="*-*-* 03:00:00" \

/usr/bin/tar -czf /var/archives/daily/syslog-$(date +\%Y-\%m-\%d).tar.gz /var/log/syslog*
```

- **Weekly Job (Friday 11:00 PM):**

```
# (Assuming the script from Q3 exists at /usr/local/bin/archive_weekly.sh)

sudo systemd-run --on-calendar="Fri *-*-* 23:00:00" /usr/local/bin/archive_weekly.sh
```

Method 2: Using Persistent systemd Timer and Service Units (Recommended)

This is the modern, standard replacement for cron jobs. It's more powerful and provides better logging and control.

A. Daily Archival Task

1. **Create the Service Unit (.service file):** This defines *what* to do.

   File: /etc/systemd/system/log-archive-daily.service

```
[Unit]

Description=Daily archive of syslog files



[Service]

Type=oneshot

ExecStart=/usr/bin/tar -czf /var/archives/daily/syslog-%I.tar.gz /var/log/syslog*
```

   - %I is a specifier that will be replaced by the timer unit with a timestamp.

2. **Create the Timer Unit (.timer file):** This defines *when* to do it.

   File: /etc/systemd/system/log-archive-daily.timer

```
[Unit]

Description=Run daily syslog archival job



[Timer]

OnCalendar=daily

# Or more specifically: OnCalendar=*-*-* 03:00:00

Persistent=true



[Install]

WantedBy=timers.target
```

3. **Enable and Start the Timer:**

```
sudo systemctl enable log-archive-daily.timer

sudo systemctl start log-archive-daily.timer
```

B. Weekly Move Task

1. **Create the Service Unit:**

   File: /etc/systemd/system/log-archive-weekly.service

```
[Unit]

Description=Weekly bundling and moving of log archives



[Service]

Type=oneshot

ExecStart=/usr/local/bin/archive_weekly.sh
```

2. **Create the Timer Unit:**

   File: /etc/systemd/system/log-archive-weekly.timer

```
[Unit]

Description=Run weekly log archival and move job



[Timer]

OnCalendar=Fri 23:00:00

Persistent=true



[Install]

WantedBy=timers.target
```

3. **Enable and Start the Timer:**

```
sudo systemctl enable log-archive-weekly.timer

sudo systemctl start log-archive-weekly.timer
```

You can check the status of all timers with systemctl list-timers.

Question 5:

How does hwclock work? What happens to the hardware clock when the device is powered off?

The **hardware clock (HWCLOCK)**, also known as the Real-Time Clock (RTC) or CMOS clock, is a physical component on a computer's motherboard.

How hwclock works:

The hwclock command is a user-space utility that provides an interface to interact with the hardware clock.

- **Independent Operation:** The RTC is an integrated circuit with its own crystal oscillator and power source (a small battery, like a CR2032 coin cell). This allows it to keep track of time independently of the main system's power state.
- **Interaction with the System Clock:**
    1. **On Boot:** The Linux kernel reads the time from the hardware clock and uses it to set the initial time for the **system clock** (the software clock maintained by the kernel).
    2. **During Operation:** The system clock runs independently. It is more precise but can "drift" over time.

3. **On Shutdown:** In a typical configuration, the hwclock --systohc command is run during the shutdown process. This takes the (often more accurate, especially if synced with NTP) time from the system clock and writes it back to the hardware clock. This keeps the RTC accurate for the next boot.

- **Common hwclock commands:**
    - sudo hwclock --show or sudo hwclock -r: Read and display the time from the RTC.
    - sudo hwclock --systohc or sudo hwclock -w: Set the RTC from the current system clock.
    - sudo hwclock --hctosys or sudo hwclock -s: Set the system clock from the RTC.

What happens when the device is powered off?

When the main power to the device is turned off, the **hardware clock continues to run**. The small battery on the motherboard provides the necessary power to the RTC chip, ensuring that it maintains the correct date and time.

This is its primary function: to preserve the time across power cycles so the system has a valid starting time when it boots up again. If the RTC battery dies, the clock will reset (often to a default date like Jan 1, 1970) every time the computer loses main power.

Question 6:

Use Chrony to set up a time synchronization service where vm1 acts as the Chrony server and vm2 synchronizes its time using vm1 as the source（do not use external time servers）.

Here's how to configure vm1 as a local NTP server and vm2 as its client using chrony. Let's assume vm1 has the IP address 192.168.1.10.

Step 1: Configure vm1（The Server）

1. **Install Chrony:**

```
sudo apt update && sudo apt install chrony # Debian/Ubuntu

# or

sudo dnf install chrony # RHEL/Fedora/CentOS
```

2. **Edit the Configuration**

   File: Open /etc/chrony/chrony.conf (or /etc/chrony.conf).

```
sudo nano /etc/chrony/chrony.conf
```

3. **Modify the Configuration:**

   - **Comment out external pools:** Find the pool or server lines that point to public NTP servers and comment them out with a #.

```
# pool 2.debian.pool.ntp.org iburst
```

- **Allow clients to connect:** Add an allow directive to specify which network or clients can sync from this server. For example, to allow any client from the 192.168.1.0/24 subnet:

```
allow 192.168.1.0/24
```

- **Enable serving time locally:** Add the local directive. This is critical. It tells chronyd to act as a time source for its clients even if it is not itself synchronized to a better time source. stratum 10 is a common choice for local, unsynchronized servers to prevent them from being preferred over real internet servers.

```
local stratum 10
```

4. **Restart and Enable Chrony:**

```
sudo systemctl restart chronyd

sudo systemctl enable chronyd
```

5. **Configure Firewall:** Allow incoming NTP traffic on UDP port 123.

```
sudo firewall-cmd --add-service=ntp --permanent

sudo firewall-cmd --reload

# Or with ufw:

# sudo ufw allow 123/udp
```

Step 2: Configure vm2 (The Client)

1. **Install Chrony:** Install it on vm2 just as you did on vm1.
2. **Edit the Configuration File:** Open /etc/chrony/chrony.conf on vm2.
3. **Modify the Configuration:**
    - **Comment out external pools:** Just like on the server, comment out all existing pool or server lines.
    - **Add vm1 as the server:** Add a new server line that points to vm1. The iburst option sends a quick burst of packets at startup to speed up the initial synchronization.

```
server 192.168.1.10 iburst
```

4. **Restart and Enable Chrony:**

```
sudo systemctl restart chronyd

sudo systemctl enable chronyd
```

Step 3: Verification

- **On vm2 (client):** Run chronyc sources. After a minute or two, you should see vm1 listed as a time source. The ^ indicates it's a server, and a * prefix means vm2 is now synchronized to it.

```
$ chronyc sources

MS Name/IP address         Stratum Poll Reach LastRx Last sample

===============================================================================

^* 192.168.1.10                10   6   377    21   +1.234us[ +2.456us] +/- 15ms
```

- **On vm1 (server):** Run chronyc clients. You should see the IP address of vm2.

```
$ chronyc clients

Hostname                      NTP Packets    Drop Int Drop Int Last

===============================================================

192.168.1.20                  5   0  6  -   0  25s
```

Question 7:

Explain in detail how the system clock works. What does it mean to synchronize the system time using NTP? What happens if the NTP server goes down? What is meant by 'time drift' in the context of NTP?

System Clock

The **system clock**, also called the software clock, is the timekeeper for the operating system. It is maintained by the kernel and is what all applications and system services use when they need to know the current time.

- **Initialization:** When a computer boots, the kernel initializes the system clock by reading the current time from the **hardware clock (RTC)**.

- **Operation:** After initialization, the system clock runs independently. It is essentially a counter that gets incremented by timer interrupts generated by the system's hardware (a quartz crystal oscillator) at a fixed frequency (e.g., HZ).
- **Inaccuracy:** This hardware oscillator is not perfectly accurate. Due to manufacturing imperfections and environmental factors like temperature, its frequency can vary slightly. This leads to the system clock running a little bit faster or slower than true time.

Time Drift

Time drift is the phenomenon where the system clock gradually becomes inaccurate because its underlying oscillator frequency is not perfect. Over hours, days, and weeks, this small error accumulates, causing the system's time to "drift" away from the actual time by seconds or even minutes. This is precisely the problem that NTP is designed to solve.

Synchronizing with NTP

NTP（Network Time Protocol） is a networking protocol designed to synchronize the clocks of computers over a network.

- **How it Works:**
    1. An NTP client (like chronyd or ntpd) is configured with the address of one or more NTP servers.
    2. The client sends a packet to the server, recording the time it sent it (T1).

3. The NTP server receives the packet (at T2) and sends a reply, including T2 and the time it sent the reply (T3).
4. The client receives the reply (at T4).
5. With these four timestamps (T1, T2, T3, T4), the client can calculate two important values:
    - **Round-trip delay:** The time it took for the packet to go to the server and back. Delay = (T4 - T1) - (T3 - T2).
    - **Time offset:** The difference between the client's clock and the server's clock. Offset = ((T2 - T1) + (T3 - T4)) / 2.
6. The client collects several such samples to get a reliable estimate of the offset.
7. Instead of abruptly changing the time (which can break applications), the client **slews** the clock. It slightly speeds up or slows down the system clock's rate until it matches the server's time.

What Happens if the NTP Server Goes Down?

When the NTP server becomes unreachable, the client software will detect this.

- The client will mark the server as unreachable and stop using it for synchronization.
- The client's clock will now be in "free-run" mode. It will continue to run based on its own hardware oscillator.

- Modern NTP clients like chrony are intelligent. They keep a record of the clock's historical drift rate in a **drift file** (e.g., /var/lib/chrony/drift). When the server is lost, chrony will continue to apply this last known correction factor to the system clock. This significantly reduces the rate of drift compared to a system with no compensation.
- However, without a live reference, the clock will inevitably begin to drift again. The accuracy will degrade over time until the NTP server becomes available again, at which point the client will re-synchronize.

Question 8:

Write a script that logs user login attempts when a wrong password is entered. This script should be implemented as a systemd service unit.

This solution involves two parts: a Bash script to monitor and log the failures, and a systemd service unit to run the script as a daemon.

Part 1: The Monitoring Script

This script will use journalctl to follow the system log in real-time and grep to filter for failed password messages.

Create the script file at /usr/local/bin/failed_login_monitor.sh:

```
sudo nano /usr/local/bin/failed_login_monitor.sh
```

Add the following content:

```
#!/bin/bash



# Define the log file where we'll store our custom messages

LOG_FILE="/var/log/failed_logins.log"



# Use journalctl to follow logs in real-time.

# We pipe this into a while loop that reads line by line.

# The grep filters for common phrases indicating a failed password attempt.

# This works for SSH, sudo, and local logins.

journalctl -f | grep --line-buffered -E "Failed password|authentication failure" | while read -r line; do

  # When a match is found, append a formatted entry to our custom log file

  echo "$(date) | FAILED LOGIN DETECTED | $line" >> "$LOG_FILE"

done
```

Make the script executable:

```
sudo chmod +x /usr/local/bin/failed_login_monitor.sh
```

- journalctl -f: Follows the system journal.
- grep --line-buffered: Ensures grep outputs each matching line immediately instead of buffering. This is crucial when piping.

- -E "Failed password|authentication failure": A regular expression to catch different types of auth failures.
- while read -r line: Reads the output from grep line by line.

Part 2: The systemd Service Unit

This service will manage the script, ensuring it runs on boot and restarts automatically if it ever crashes.

Create the service unit file at /etc/systemd/system/failed_login_monitor.service:

```
sudo nano /etc/systemd/system/failed_login_monitor.service
```

Add the following content:

```
[Unit]

Description=Monitors for failed login attempts and logs them

After=network.target



[Service]

ExecStart=/usr/local/bin/failed_login_monitor.sh

Restart=always

RestartSec=10

User=root

Group=root



[Install]

WantedBy=multi-user.target
```

- Description: A human-readable description of the service.
- ExecStart: The command to run to start the service.
- Restart=always: Tells systemd to automatically restart the service if it stops for any reason.
- WantedBy=multi-user.target: Ensures the service starts at boot when the system reaches a multi-user state.

Part 3: Enable and Start the Service

1. **Reload the systemd daemon** to make it aware of the new service file:

```
sudo systemctl daemon-reload
```

2. **Enable the service** to start on boot:

```
sudo systemctl enable failed_login_monitor.service
```

3. **Start the service** immediately:

```
sudo systemctl start failed_login_monitor.service
```

4. **Check its status**:

```
sudo systemctl status failed_login_monitor.service
```

Now, any failed login attempt on the system will be logged to /var/log/failed_logins.log. You can test this by trying to sudo with a wrong password or failing an SSH login. Then check the log file with tail -f /var/log/failed_logins.log.

Question 9:

Set up an rsyslog server on vm1 and configure it to receive and store logs from /var/log, journalctl, and dmesg from other systems.

This creates a centralized logging server. We'll configure vm1 to be the rsyslog server and show how to configure a client to send its logs.

Step 1: Configure vm1 (The rsyslog Server)

1. **Install rsyslog** (it's usually installed by default):

```
sudo apt update && sudo apt install rsyslog
```

2. **Edit the configuration file** /etc/rsyslog.conf:

```
sudo nano /etc/rsyslog.conf
```

3. **Enable Log Reception:** Find the #### MODULES #### section and uncomment the lines for TCP log reception. TCP is more reliable than UDP.

```
# provides TCP server functionality

module(load="imtcp")

input(type="imtcp" port="514")
```

4. **Create a Template for Storing Remote Logs:** Go to the #### TEMPLATES #### section (or create it if it doesn't exist). Add a template that will create log files based on the hostname of the client machine. This keeps logs organized.

```
# Template to store remote logs in /var/log/remote/<hostname>.log

$template RemoteLogs,"/var/log/remote/%HOSTNAME%.log"
```

5. **Create a Rule to Use the Template:** Add a rule that directs all incoming messages from any host (that isn't the local host) to be written using your new template. This should go before the local logging rules (like *.* /var/log/syslog).

```
# Log all messages from remote hosts to the RemoteLogs template

:fromhost-ip, !isequal, "127.0.0.1" ?RemoteLogs

& stop
```

- :fromhost-ip, !isequal, "127.0.0.1": This filter selects messages that are not from the local machine.
- ?RemoteLogs: This applies the RemoteLogs template.
- & stop: This tells rsyslog to stop processing this message further, so it doesn't also get written to the server's local log files.

6. **Restart rsyslog:**

```
sudo systemctl restart rsyslog
```

7. **Open Firewall Port:** Allow incoming traffic on TCP port 514.

```
sudo firewall-cmd --add-port=514/tcp --permanent

sudo firewall-cmd --reload

# Or with ufw:

# sudo ufw allow 514/tcp
```

Step 2: Configure a Client Machine (e.g., vm2)

1. **Edit the client's /etc/rsyslog.conf file:**

```
sudo nano /etc/rsyslog.conf
```

2. **Add a Forwarding Rule:** Go to the very end of the file and add a rule to forward all log messages (*.*) to the rsyslog server (vm1). Use @@ for TCP.

```
# Forward all logs to the central rsyslog server vm1 via TCP

*.* @@vm1:514
```

3. **Restart rsyslog on the client:**

```
sudo systemctl restart rsyslog
```

How This Covers /var/log, journalctl, and dmesg:

- **/var/log:** The *.* @@vm1:514 rule on the client forwards every message that rsyslog processes. Since rsyslog is what populates files like /var/log/syslog and /var/log/auth.log, you are effectively forwarding their content.
- **journalctl:** On modern systems, rsyslog uses the imjournal module by default to read messages directly from the systemd journal. The journal is the central collection point for logs from all services. By forwarding rsyslog's messages, you are forwarding the journal's content.
- **dmesg:** Kernel ring buffer messages (what dmesg shows) are also written to the systemd journal. Because rsyslog reads from the journal, these kernel messages are included and will be forwarded to the server.

After a few moments, you should see logs from the client appearing on vm1 in a file named /var/log/remote/<client_hostname>.log.