Section 1: Sending Emails via Terminal

Overview

This section primarily focuses on setting up our Linux environment to send automated emails from the terminal. It's a crucial step in creating a script that will later notify an admin when specific events occur on the system, such as excessive `sudo` attempts by a user.

The key concepts in this section include:

- Using terminal commands over a GUI for automation.
- Setting up Gmail for sending emails from the terminal.
- Using mail transfer agents (MTAs) such as **Postfix** to handle email delivery.

1.1. Importance of Terminal Commands for Automation

The section starts by emphasizing the significance of terminal-based automation for software engineers. GUIs have limitations when it comes to repetitive or scheduled tasks, making the terminal a more powerful tool for developers who need fine-grained control over their system. Terminal automation allows:

- Easier scheduling of tasks via cron jobs.
- Writing shell scripts for repeated tasks.
- Interacting directly with system logs and processes.

1.2. Using WSL for Windows Users

For Windows users who do not have access to a native Linux environment, the document suggests using **Windows Subsystem for Linux (WSL)**. WSL enables you to run a Linux distribution directly on Windows without the need for dual booting or virtual machines.

- **Steps for WSL Setup**:
    1. Enable WSL via the `Turn Windows Features on or off` menu.
    2. Install your preferred Linux distribution from the Microsoft Store (e.g., Ubuntu).
    3. Use terminal commands just like in a native Linux environment.

1.3. Setting Up App Passwords for Gmail

Since Gmail requires extra security measures to allow third-party applications to send emails (like terminal-based programs), you need to set up an **App Password**. Google does not allow traditional login credentials for external apps due to security concerns, so the document provides instructions on how to generate a secure password for email usage via terminal commands.

- **Steps to Generate an App Password**:
  1. Sign in to your Gmail account.
  2. Go to **Account Settings** > **Security** > **App Passwords**.
  3. Generate a new app password specifically for terminal usage.
  4. Note down this password as it will be used in your script for sending emails.

This step ensures that you can send emails from the terminal using Gmail's SMTP server without exposing your primary Gmail password.

1.4. Installing Required Packages (mailutils)

To send emails from the terminal, you need to install the appropriate software packages. In this case, **mailutils** is used, which provides the `mail` command-line utility to send emails.

- **Installation Command**:

  ```
  sudo apt install mailutils
  ```

  This installs the necessary utilities for interacting with your mail server (in this case, Gmail) and sending messages from your scripts or terminal commands.

1.5. Configuring Postfix

Postfix is a widely-used mail transfer agent (MTA) that handles the delivery of emails from your system. The section explains how to configure Postfix to work with Gmail's SMTP server for email delivery.

- **Steps for Configuring Postfix**:

1. When installing **mailutils**, you'll be prompted to configure Postfix.
2. Choose **"Internet Site"** when asked for the configuration type.
3. Set the **System Mail Name** as your server's name or domain (or just leave it as the default).

Postfix handles the actual email sending process, connecting to Gmail's servers on your behalf.

1.6. Configuring Postfix for Gmail (SMTP Setup)

To configure Postfix to work with Gmail, follow these steps to adjust the SMTP settings in the Postfix configuration file:

1. **Edit the Postfix Configuration**:
   Open the `main.cf` file:

   ```
   sudo nano /etc/postfix/main.cf
   ```

2. **Add Gmail SMTP Settings**:
   Add or modify the following lines in the `main.cf` file:

   ```
   relayhost = [smtp.gmail.com]:587
   smtp_use_tls = yes
   smtp_sasl_auth_enable = yes
   smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
   smtp_sasl_security_options = noanonymous
   ```

3. **Configure Gmail Login Credentials**:
   Create or modify the `sasl_passwd` file to store your Gmail SMTP username and app password:

   ```
   sudo nano /etc/postfix/sasl_passwd
   ```

   Add the following line, replacing `your_email@gmail.com` with your Gmail address and `app_password` with the app password you generated earlier:

   ```
   [smtp.gmail.com]:587 your_email@gmail.com:app_password
   ```

4. **Secure the Password File**:
   Change the file permissions to make it accessible only by Postfix:

```
sudo chmod 600 /etc/postfix/sasl_passwd
```

5. **Hash the Password File**:
   Convert the `sasl_passwd` file into a `.db` file that Postfix can read:

   ```
   sudo postmap /etc/postfix/sasl_passwd
   ```

6. **Restart Postfix**:
   After making these changes, restart Postfix to apply the configuration:

   ```
   sudo systemctl restart postfix
   ```

1.7. Testing Email Sending from Terminal

Once Postfix is configured, you can test sending an email from the terminal to confirm everything is working correctly. Use the following command to send a test email:

```
echo "This is a test email." | mail -s "Test Subject" recipient@example.com
```

- `echo "This is a test email."` writes the body of the email.
- `-s "Test Subject"` defines the subject of the email.
- `recipient@example.com` is the email address where the email will be sent.

If everything is configured correctly, the recipient should receive the test email within a few minutes.

Summary

- **Objective**: The goal of this section is to set up the ability to send automated emails from the terminal using Gmail.
- **Key Steps**:
  1. Install necessary packages (mailutils).
  2. Configure Postfix as an MTA with Gmail's SMTP server.
  3. Generate an app password from Gmail for secure login.

4. Test the setup by sending an email from the terminal.

Section 2: Initiating the Process of Sending Mail Based on Conditions

---

Overview

Section 2 delves into applying the environment we set up in the previous section to initiate an automated process that triggers email alerts when specific conditions are met. The section focuses on monitoring user activities related to the use of the `sudo` command and sending an email notification when a threshold of failed `sudo` attempts is reached. This is where the automation aspect of the project comes into play, as we'll write and execute a shell script to manage this process.

2.1. Creating a Low-Privilege User

The first task in this section is to create a low-privilege user for testing purposes. This user will be monitored for their use of the `sudo` command. A low-privilege user is one who does not have administrative rights, and the system will log any failed attempts by this user to execute privileged commands.

Command to Create a New User:

```
sudo adduser bob
```

Here, the new user "bob" is created. The `adduser` command prompts you to set a password and provides other details like full name, which can be left blank if not necessary.

After creating the user, ensure that "bob" does not have `sudo` privileges. You can confirm this by checking the `/etc/sudoers` file, which lists the users and groups allowed to use `sudo`.

2.2. Understanding the Sudoers File and Privileges

The `sudoers` file controls which users are allowed to run commands as a superuser or another user. This file can only be edited safely using the `visudo` command, which locks the file to prevent syntax errors that could lock you out of `sudo` privileges entirely.

- **Viewing the Sudoers File**:
  You can view or edit the `sudoers` file using the command:

  ```
  sudo visudo
  ```

  In the file, you'll see entries that define which users or groups have superuser privileges. If the user "bob" is not listed, they will not be able to use the `sudo` command, and failed attempts will be logged.

2.3. Writing the Shell Script

Now, the core part of this section is writing a shell script that monitors the logs for failed `sudo` attempts by users like "bob" and sends an email notification when a specific condition (threshold of attempts) is met.

Key Components of the Shell Script:

- **Log Monitoring**:
  The log file `/var/log/auth.log` stores information about authentication attempts, including failed `sudo` attempts. The script will use this log file to track and count how many times a user unsuccessfully tries to use `sudo`.
- **Threshold Condition**:
  A threshold (for example, 3 failed attempts) is defined. When the user's failed `sudo` attempts exceed this threshold, the system triggers an action (sending an email).
- **Sending Email**:
  The email functionality, which you set up in Section 1, is now incorporated into the script. When the threshold is reached, the script will send an email to the admin.

Example Shell Script:

```
#!/bin/bash

# Define the threshold for sudo attempts
THRESHOLD=3
```

```
# Path to the sudo log file
LOG_FILE="/var/log/auth.log"

# Admin email address
ADMIN_EMAIL="admin@gmail.com"

# Extract low-privilege users (users with UIDs >= 1000 but not in sudoers)
LOW_PRIV_USERS=$(awk -F: '$3 >= 1000 && $3 != 65534 {print $1}' /etc/passwd)

# Loop through the list of low-privilege users
for USER in $LOW_PRIV_USERS; do
  # Count the number of sudo attempts by the user
  SUDO_COUNT=$(grep -c "sudo:.*${USER}" $LOG_FILE)

  # Check if the sudo count exceeds the defined threshold
  if [ $SUDO_COUNT -gt $THRESHOLD ]; then
    # Define the subject and body of the email
    SUBJECT="Excessive sudo attempts by $USER"
    BODY="The user $USER has attempted to use sudo $SUDO_COUNT times."

    # Send an email to the admin
    echo "$BODY" | mail -s "$SUBJECT" "$ADMIN_EMAIL"
  fi
done
```

- **How the Script Works**:
  1. The script sets a threshold of `sudo` attempts (e.g., 3).
  2. It retrieves all non-root users (those with a UID >= 1000) from the `/etc/passwd` file.
  3. It monitors the `/var/log/auth.log` file for each user's `sudo` activity using `grep`.
  4. If a user exceeds the threshold of `sudo` attempts, the script sends an email notification to the admin using the `mail` command.

2.4. Breakdown of the Shell Script

1. **Threshold**:
   The variable `THRESHOLD` defines the number of failed sudo attempts allowed before the admin is notified. This is set to 3 in the example script, but you can adjust it based on your requirements.
2. **Log File**:
   The variable `LOG_FILE` is set to `/var/log/auth.log`, where authentication attempts are

logged, including all uses of the `sudo` command. This file is crucial for tracking `sudo` attempts.

3. **Admin Email**:
   The `ADMIN_EMAIL` variable contains the email address to which notifications will be sent. This email address was previously set up in Section 1 when you configured Gmail for sending emails from the terminal.
4. **User Extraction**:
   The line `LOW_PRIV_USERS=$(awk -F: '$3 >= 1000 && $3 != 65534 {print $1}' /etc/passwd)` extracts all users with a User ID (UID) greater than or equal to 1000, which corresponds to normal system users. This excludes system users with UIDs lower than 1000 and "nobody" (UID 65534).
5. **Counting Failed Attempts**:
   For each user, the script uses `grep` to search for failed sudo attempts in the log file. The `grep` command looks for lines that match the pattern `sudo:.*${USER}`, which corresponds to sudo attempts by the user. The `-c` flag counts the number of occurrences.
6. **Condition Check**:
   If the count of failed `sudo` attempts (`SUDO_COUNT`) exceeds the `THRESHOLD`, the script sends an email alert. The email body contains information about the user and the number of failed attempts.
7. **Sending Email**:
   The script uses the `mail` command to send an email. The subject (`SUBJECT`) and body (`BODY`) of the email are dynamically created to reflect the user who triggered the alert and the number of attempts.

2.5. Testing the Shell Script

Once the script is written, you should test it to ensure it functions as expected. Here's how you can test it:

1. **Run the Script Manually**:
   You can manually run the script from the terminal to ensure that it correctly monitors the log file and sends email alerts. Use the following command:

   ```
   ./script_name.sh
   ```

2. **Check the Logs**:
   You can simulate failed `sudo` attempts by using a low-privilege user to try running commands with `sudo`. Then, check the log file using:

```
tail -f /var/log/auth.log
```

This allows you to see the real-time output of authentication attempts, ensuring the script is accurately counting failed attempts.

3. **Confirm Email Delivery**:
   Ensure that the email notifications are being sent to the admin's email address. Check both the logs and your Gmail account to confirm the email was delivered.

Summary

- **Objective**: The goal of Section 2 is to set up a monitoring system that tracks `sudo` command usage by low-privilege users and sends email notifications when a defined threshold is exceeded.
- **Steps**:
  1. Create a new low-privilege user for testing.
  2. Monitor failed `sudo` attempts in the system logs.
  3. Write a shell script that checks for failed attempts, compares the count to a threshold, and sends an email when the threshold is exceeded.

Section 3: Automating the Process with Cron Jobs

Overview

Section 3 expands on the previous sections by introducing automation using **cron jobs**. After configuring the email system (Section 1) and setting up the monitoring script for tracking `sudo` attempts (Section 2), the final step is to automate the entire process to run periodically without manual intervention. This section focuses on using the cron utility to schedule the shell script from Section 2 to execute at specified intervals.

3.1. What is Cron?

Cron is a time-based job scheduler in Unix-like operating systems. It allows users to schedule tasks (called cron jobs) to run automatically at specified intervals, be it minutes, hours, days, or even months. These jobs are typically background processes, and they execute commands or scripts as per the schedule defined by the user.

In this section, cron will be used to run the monitoring script periodically to check for failed `sudo` attempts and send email alerts when the predefined conditions are met.

3.2. Writing the Cron Job Entry

Cron syntax follows a specific pattern of five fields followed by the command or script to execute. Each field represents a time unit:

```
* * * * * command
- - - - -
| | | | |
| | | | |_____ Day of the week (0-7, where 0 and 7 are Sunday)
| | | |_______ Month (1-12)
| | |_________ Day of the month (1-31)
| |___________ Hour (0-23)
|_____________ Minute (0-59)
```

Each of these fields can have a specific value or be replaced with an asterisk (`*`) to indicate "every" unit of that time. For example, `* * * * *` means the task will run every minute, whereas `0 0 * * *` means the task will run at midnight every day.

3.3. Scheduling the Monitoring Script

In this section, you are asked to run the script **6 times a day** at the following times:

- **12:00 AM**
- **4:00 AM**
- **8:00 AM**
- **12:00 PM**
- **4:00 PM**
- **8:00 PM**

The corresponding cron expression for this schedule is:

```
0 0,4,8,12,16,20 * * * /path/to/your/script.sh
```

This command will execute the script every day at the specified times.

- **Explanation**:
  - `0`: The script will run at the start of the hour (exactly on the hour).
  - `0,4,8,12,16,20`: The script will run at 12:00 AM, 4:00 AM, 8:00 AM, 12:00 PM, 4:00 PM, and 8:00 PM.
  - `* * *`: These represent "every day of the month," "every month," and "every day of the week."

Steps to Add a Cron Job:

1. **Open the Cron Table (Crontab)**:
   The cron table (crontab) is a file that stores the schedule for cron jobs. You can open it using the following command:

   ```
   crontab -e
   ```

2. **Add the Cron Entry**:
   At the end of the crontab file, add the following line:

   ```
   0 0,4,8,12,16,20 * * * /path/to/your/script.sh
   ```

   Replace `/path/to/your/script.sh` with the actual path to your shell script. This line ensures that the monitoring script will run at the specified times every day.
3. **Save and Exit**:
   After adding the cron entry, save the file and exit the editor. Cron will automatically pick up the new job and start executing it at the next scheduled time.

3.4. Verifying the Cron Job is Running

After scheduling the cron job, it's important to verify that it works as expected.

Methods to Verify Cron Job Execution:

1. **Check the Cron Logs**:
   Cron maintains logs of all jobs it runs. You can check the logs to see whether your script is being executed by cron.

   ```
   grep CRON /var/log/syslog
   ```

   This will display the cron job executions, including timestamps and the script name, allowing you to confirm that the job is running at the specified times.
2. **Use a Debugging Statement in Your Script**:
   You can add a line in your script to log the execution time to a file for debugging purposes. For example, add the following line to the script:

   ```
   echo "Script executed on $(date)" >> /path/to/logfile.txt
   ```

   Each time the script runs, this line will append the current date and time to `logfile.txt`. By checking this file, you can confirm that the cron job is executing the script on schedule.
3. **Confirm Email Notifications**:
   If the script is functioning correctly, you should receive email notifications when the threshold of failed `sudo` attempts is exceeded. You can check your inbox to verify that the emails are being sent at the correct times.

3.5. Managing Cron Jobs

List Existing Cron Jobs:

To view all cron jobs scheduled for the current user, use:

```
crontab -l
```

Remove a Cron Job:

To remove a cron job, open the crontab file using `crontab -e` and delete the corresponding line, or comment it out by adding a `#` at the beginning of the line.

3.6. Potential Issues and Troubleshooting

While working with cron jobs, you may encounter certain issues. Here are some common issues and troubleshooting steps:

- **Path Issues**:
  Sometimes, cron jobs fail because the environment in which they run does not have access to certain paths (e.g., the script may work when run manually, but not from cron). You can resolve this by using absolute paths in your script and ensuring that all necessary environment variables are set in the script.
- **Permissions**:
  Ensure that the script has the correct permissions to be executed by the cron job. You can check and set the permissions using:

  ```
  chmod +x /path/to/your/script.sh
  ```

- **Email Alerts for Cron Failures**:
  By default, cron sends an email to the user when a job fails (if the system is configured for email). If you want to redirect the error output to a file instead, you can modify the cron job entry as follows:

  ```
  0 0,4,8,12,16,20 * * * /path/to/your/script.sh 2>> /path/to/error_log.txt
  ```

  This will capture any errors in `error_log.txt` for further inspection.

Summary

- **Objective**: The goal of Section 3 is to automate the monitoring and alert system created in Section 2 using cron jobs.
- **Steps**:
  1. Use cron to schedule the shell script to run 6 times a day at specified intervals (every 4 hours starting at midnight).
  2. Add the cron job using the `crontab -e` command.
  3. Verify that the cron job is working by checking the cron logs or adding debugging statements to the script.
  4. Handle potential issues like path problems, permissions, and error logging.