HyperionDev

Task Scheduling and Log Monitoring with Cron

Task

Visit our website

Introduction

In cybersecurity, many defensive activities must be performed repeatedly and consistently to maintain a secure environment. These include log reviews, integrity checks, malware scans, permission audits, backup verification, and system updates. Performing these tasks manually is not only inefficient but also increases the likelihood of human error. Cron allows security professionals to automate routine tasks so they occur reliably at the correct time, even when no one is actively monitoring the system.

Task scheduling strengthens operational resilience by ensuring that essential activities continue uninterrupted. For example, if a system administrator forgets to rotate logs or run update checks, vulnerabilities may accumulate quietly in the background. By automating these tasks with cron, organisations reduce risk and maintain a predictable, well-managed system environment.

Cron as a Monitoring Tool

Although cron is primarily known for task automation, it also plays a valuable role in system monitoring. Many logs and operational metrics follow daily, hourly, or periodic patterns. Cron can be used to check these patterns and alert administrators when something unexpected occurs. Because cron executes in a predictable, controlled manner, it is an ideal tool for creating lightweight monitoring mechanisms without installing additional software.

For example, cron can run a script every 10 minutes to review authentication logs for suspicious login attempts. If the script detects repeated failed logins or login activity from strange IP addresses, it can append an entry to a security log, send an alert, or trigger another automated response. This type of scheduled monitoring is especially useful on servers that must operate continuously and require constant oversight.

Automating Log Rotation and Storage

Linux systems continuously generate logs, including kernel logs, system logs, authentication logs, and application logs. If these logs are not rotated and archived regularly, they can consume disk space and eventually cause system failures. Cron provides a structured way to automate log rotation using custom scripts.

Copyright © 2025 HyperionDev. All rights reserved.

A scheduled log rotation script can:

- Create a compressed archive of old logs
- Move archived logs to a secure storage location
- Clear or recreate log files with correct permissions
- Ensure that log files remain readable for auditing

This prevents log overflow and ensures that important information remains available for forensic analysis and incident response.

Running Security Scans on a Schedule

Many security tools rely on routine execution to detect threats effectively. With cron, system administrators can schedule:

- File integrity monitoring tools like **AIDE**
- Virus scans with **ClamAV**
- Vulnerability scans
- Backup verification routines
- Permission audits on sensitive directories

By automating these activities, cron ensures that potential threats are identified as early as possible, without requiring constant human intervention. This is especially important in security operations where rapid detection can significantly limit the impact of an attack.

Automated Alerting with Cron Scripts

Cron-driven scripts can also be configured to send alerts when certain conditions are met. For instance, a scheduled script might check:

- Disk usage
- CPU or memory consumption
- Suspicious file modifications

Copyright © 2025 HyperionDev. All rights reserved.

- Unusual outbound connections
- New user accounts created without authorisation

If any anomalies are detected, the script could automatically email the administrator or send a message to a monitored log file. This gives cybersecurity teams immediate visibility into emerging issues, supporting faster triage during incidents.

Enhancing Incident Response Through Automation

During a cyber incident, time is critical. Cron can support incident response by enabling automated evidence collection at regular intervals. Scheduled scripts can:

- Capture snapshots of running processes
- Record network connections
- Archive critical logs
- Verify system integrity
- Take system-state backups

Even if an attacker attempts to hide their tracks, scheduled monitoring ensures that the system continues to collect evidence over time.

Additionally, post-incident recovery tasks—such as restarting essential services, verifying configurations, or restoring file permissions—can be automated to run after reboot or on a predefined schedule.

Integrating Cron with Broader Security Practices

Cron does not replace enterprise-level monitoring or orchestration platforms, but it complements them by providing lightweight, scriptable, and reliable scheduling capabilities. In environments where system resources are limited or full-scale monitoring tools are unavailable, cron becomes an essential component of the security toolkit.

By combining cron with Bash scripting, administrators can build powerful automated workflows that detect issues early, enforce consistent security practices, and maintain long-term system stability.

Copyright © 2025 HyperionDev. All rights reserved.

Cron is a daemon (background process) that runs tasks at scheduled times. A cron job is a Linux utility that schedules tasks for specified intervals. Cron jobs are often used for maintenance tasks, such as backing up databases, sending emails, and running system updates. By the end of this task, you will have a clearer understanding of how processes work in the background within Linux, and gain an appreciation for any processes that are running and scheduling new tasks using cron.

Command notation

Let's first wrap our heads around some basic notational conventions.

This task will use `$` and `#` symbols before commands. The `$` symbol represents commands that are run by a regular user, meaning you don't need any special privileges to execute them. In contrast, the `#` symbol is used to show that the command must be run with root (or administrative) privileges. To execute commands as the root user, you can either prefix them with `sudo` to temporarily gain elevated privileges, or you can switch to the root account entirely by logging in as the root user using the command `su root`.

For example, the command `echo "Hello, world!"`, which prints `"Hello, world!"` to the terminal, can be run by a regular user and would be written as `$ echo "Hello, world!"`. However, if you want to update the system package list, a command such as `apt-get update` must be executed with root privileges. This would be written as `# apt-get update`, indicating that you need to run it as the root user. If you are not logged in as root, you can execute it with sudo, such as `sudo apt-get update`.

Take note

You must be **very careful** before running any commands as the root user. Very few tools will stop the root account from doing things that will break the system. With great power comes great responsibility!

Copyright © 2025 HyperionDev. All rights reserved.

Process management

There are two kinds of processes: **foreground** and **background** processes.

Foreground processes

A foreground process is one that is executed by the user and is interactive. While such a process is running you cannot start a new process from the same terminal.

Let's take a closer look at what happens if you type what's just below into your terminal:

```
$ ls -la
```

In this example, the `ls -la` command runs in the foreground. The terminal waits for the command to finish listing the files before you can type another command.

Background processes

A background process is **non-interactive** and is usually initiated by the system or another process. A significant number of the processes running on a computer at any given time are background processes, also referred to as **daemons**. In Linux, all daemons are started by the init daemon at boot, which has a PID (process ID) of 1 and initiates all the processes required for the system to work.

systemd

systemd is the init system used by most Linux distributions today, with the exception of a few, such as **Slackware**. Once the Linux kernel completes its initial boot process, it hands control over to systemd. As the first process to run, systemd is responsible for initialising and managing the system's processes.

One of the key features of systemd is its ability to start processes in parallel, significantly improving boot times. These processes are managed according to configuration files known as unit files. These files are in a set of paths:

- **/lib/systemd/system** has OS default configuration files.
- **/etc/systemd/system** has configuration files that override the default ones.
- **/run/systemd/system** has generated configuration files that override the installed ones.

Copyright © 2025 HyperionDev. All rights reserved.

Inside these folders, there are files with many different file extensions. Everything is controlled and supervised by systemd, and the extensions encode their types in the following ways:

- ***.service** is a process.
- ***.device** is a system device inside **/sys**, and these devices provide interfaces to kernel data structures.
- ***.mount** is a file system mount point.
- ***.automount** is a file system automount point.
- ***.swap** is the swap device or file. The swap device is virtual memory, and is functionally equivalent to the page file in Windows.
- ***.path** is a path used for path-based activation. In these files, paths are monitored to watch for changes to them and their contents.
- ***.socket** describes a network and/or IPC socket for socket-based activation.
- ***.timer** describes a timer for timer-based activation.
- ***.slice** manages resources with control groups (cgroups), a kernel feature that organises processes into hierarchical groups where each layer has various types of resources that can be limited and monitored.
- ***.scope** is created programmatically with systemd's bus interfaces to manage a set of system processes.
- ***.target** groups unit configuration files to create synchronisation points during start-up. When a target is reached, all of its individual unit tasks have been completed.

During init, systemd tries to reach **/lib/systemd/system/default.target**, which in most cases is the graphic desktop target.

Some common commands

There are a multitude of commands you can use to manage processes. Here are a handful of popular ones:

- **`ps`:** Shows an instantaneous state of some of the processes that are running under your account, with their PIDs, which teletype they are running on, and what their command is. If you want an interactive view of all running processes, use `top` or `htop`.

Copyright © 2025 HyperionDev. All rights reserved.

- `kill`**:** Sends one of two signals to processes. By default, SIGTERM is sent, which gives processes a chance to do any cleanup before terminating execution. If they are not responsive, then SIGKILL may be sent, which orders the process to stop immediately and gives it no chance to do any cleanup.
- `batch`**:** Reads commands from **stdin** or a specified file which are to be executed at a later time.
- `systemctl`**:** Is used to manage the systemd system and service manager, and can be used in tandem with its `start`, `stop`, `restart`, and other commands to manage the execution state of services. For example, if you wanted to kill the dbus daemon and disable its service, you would use the following commands:

```
# systemctl stop dbus
# systemctl disable dbus
```

However, if you only wanted to restart the dbus daemon, you would use this command:

```
# systemctl restart dbus
```

Everything this command can do is extensively detailed in its **man page**, which is worth reading. If you'd like a quick glance at some of the most common commands, have a look at this article outlining the **top 50 Linux commands**.

Cron

Cron is a job-scheduling utility, analogous to the Task Scheduler in Windows systems, but with a much simpler syntax. The **crond** daemon provides the cron functionality, and reads the crontab (cron table) to run predefined scripts.

Crontabs specify the schedule for running particular commands and scripts. Each user on the system can have their own crontab, and these are stored in **/var/spool/cron/crontabs**. To enable a user to create and manage their own cron jobs, their username must be listed in the **/etc/cron.allow** file. You can quickly add a user to this file with the following command:

```
# echo [username] >> /etc/cron.allow
```

To edit your crontab, run this command:

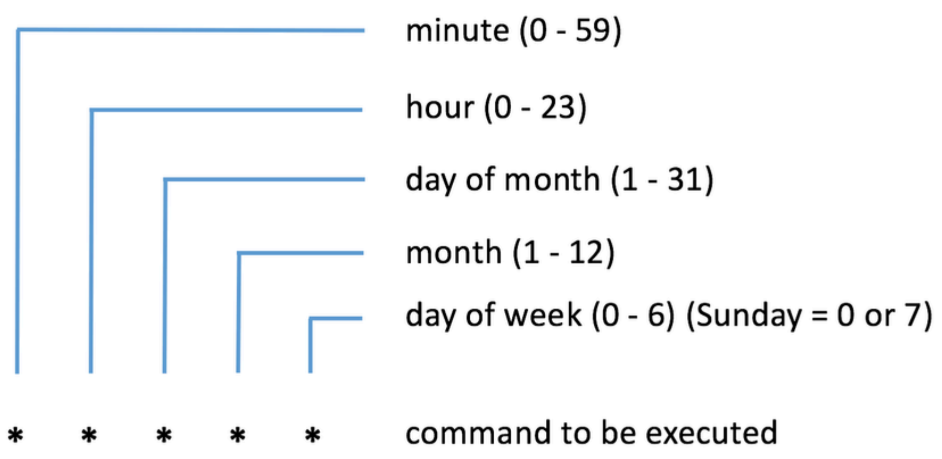
```
$ crontab -e
```

Copyright © 2025 HyperionDev. All rights reserved.

Cron jobs have the following syntax:

```
* * * * * /path/to/your/script.sh
```

From left to right, the asterisks represent **minutes** (0–59), **hours** (0–23), **days of the month** (1–31), **months** (1–12 or January to December), and **weekdays** (0–6 or Sunday to Saturday). The image below should make this a little clearer:



Mapping asterisks to time (Cocosa, 2022)

Cron job setup

1. Write a script:

   First, create a script that performs the task you want to automate. This could be any series of commands you wish to run automatically. For example, you might write a script that logs a message with the current date and time.

   ```
   #!/bin/bash
   echo "This is a log entry recorded at $(date)" >>
   /home/user/logs/cron_log.txt
   ```

   After creating the script, make sure it's executable by setting the appropriate permissions.

   ```
   chmod +x /home/user/logs/scripts/log_message.sh
   ```

2. Create or edit the crontab file:

   Next, you need to configure when and how often your script should run. This is done by editing the crontab file, which schedules tasks to be executed automatically.

Copyright © 2025 HyperionDev. All rights reserved.

Open the crontab configuration file by entering the following command:

```
crontab -e
```

3. Create the cron job:

   In the crontab file, add a new line to set up your cron job. This line will specify the timing and frequency for running your script. To run the script **log_message.sh** every day at 3:30 p.m., add this line to the crontab file:

```
30 15 * * * /home/user/logs/log_message.sh
```

   The cron job will execute at the 30th minute of each hour, and will run daily at 3 p.m. (15:00). The asterisks (*) mean the job will run every day of the month and every month. Additionally, it will run every day of the week. The path `/home/user/log_message.sh` specifies where the script to be run is located. Be sure to use the complete, absolute path to ensure the script is correctly located and run.

Cron job example

Let's say that we work a job on Saturdays and we leave for work at 9 a.m., but as we leave, we wish to run a script that does something for us.

First, let's create **all_the_way_to_work.sh**:

```
#!/bin/bash

echo `date` >> unbelievable.txt
echo "[A man inside an egg costume is seen inside a car.]" >>
unbelievable.txt
echo "AL: I can't believe I have to drive all the way to work, on a Saturday.
All the way to work!" >> unbelievable.txt
echo "[The car drives roughly 50 metres across a six-lane street from a
high-rise apartment building to a toy store, and double parks in its parking
lot.]" >> unbelievable.txt
```

This script will complain on our behalf about driving to work on a Saturday morning. We need to make sure that the script is executable:

```
$ chmod +x all_the_way_to_work.sh
```

Copyright © 2025 HyperionDev. All rights reserved.

Now, we will add it to the crontab using this format:

```
0 9 * * 6 all_the_way_to_work.sh
```

This script will now run every Saturday at 9 a.m., exactly when we leave for work.

Phew! That was a lot, but you got through. Don't worry if cron jobs feel a little unfamiliar for now. Over time, they will become second nature to you.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select "Request Review", the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you've done that, feel free to progress to the next task.

Copyright © 2025 HyperionDev. All rights reserved.

The following tasks will test your ability to schedule jobs using cron.

Auto-graded task 1

1. Write a cron job that restarts the NetworkManager service every day at 4 p.m.
   - **Hint:** Use `systemctl`.
2. Provide a screenshot of the command needed.
3. Save and submit this screenshot as **netman_job.jpg** to your task folder.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Auto-graded task 2

You decide to play a fun prank on your friend by making their computer restart every hour on Friday the 13th.

1. Write a script that achieves this (when testing your script, you can change the date).
   - **Hint**: There are several methods to achieve this. You could use `systemctl`, or use the appropriate scheduling commands. If you need to locate these commands, use the `whereis` command.
   - **Final hint:** You may need to alter your default normal privileges.
2. Provide a screenshot of the output of your alterations and the commands needed to execute this prank.
3. Save and submit your work as **prank_job.jpg** to your task folder.

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Copyright © 2025 HyperionDev. All rights reserved.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Reference list

Cocosa, J. (2022, February 23). *How to setup cron to automate your commands*. Nucleio Information Services. **https://www.nucleiotechnologies.com/how-to-setup-cron-to-automate-your-commands/**

Copyright © 2025 HyperionDev. All rights reserved.