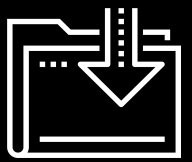
cron and Scheduled Jobs

5.2

Cybersecurity

Archiving and Logging Data Day 2



Class Objectives

By the end of today's class, you will be able to:

Schedule regular jobs for individual users with crontab.

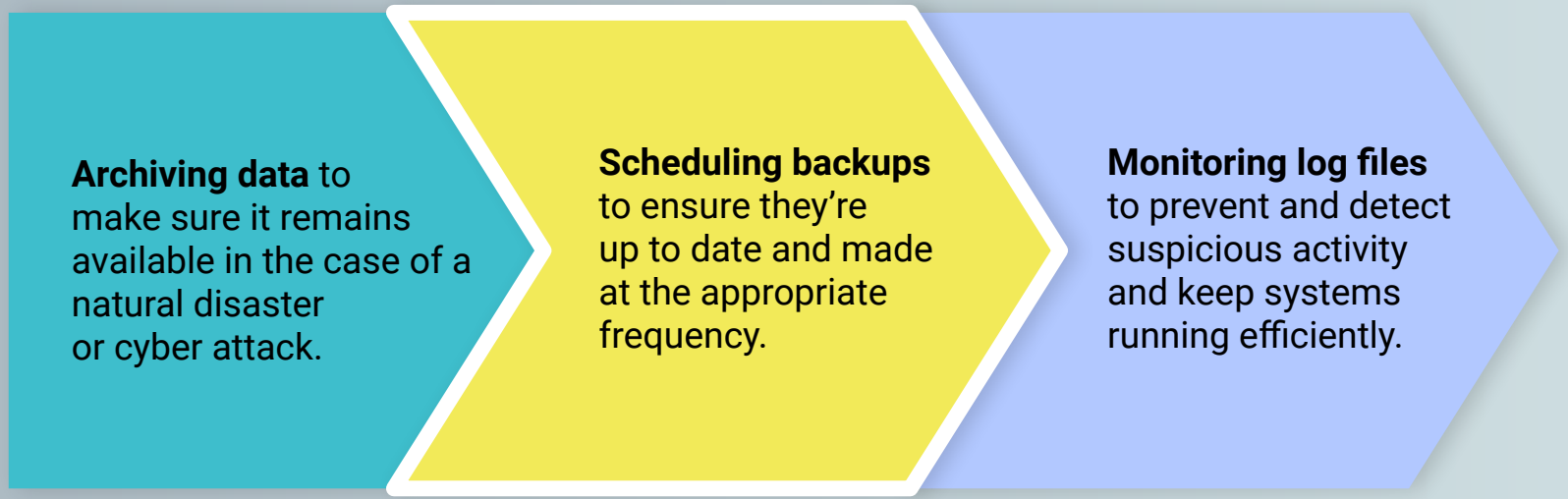
Write simple scripts for maintenance and security tasks.

Use `cron` to automate the execution of security scripts to perform maintenance on a regular basis.

Learn how to defend against attacks that use `cron`.

Scheduling Backups

Today, we'll learn how to write scripts and use a tool called `cron` to automate many of the tasks performed in the last class.



Automating

Automating a series of tasks takes two forms:

Scripts are files that contain multiple commands. The commands are executed by calling the script name.

Scheduled jobs run commands or scripts at specific, designated times.

Using Scripts in a Professional Context

Sysadmins use scripts and scheduled jobs in their workflow.

Fix an issue.

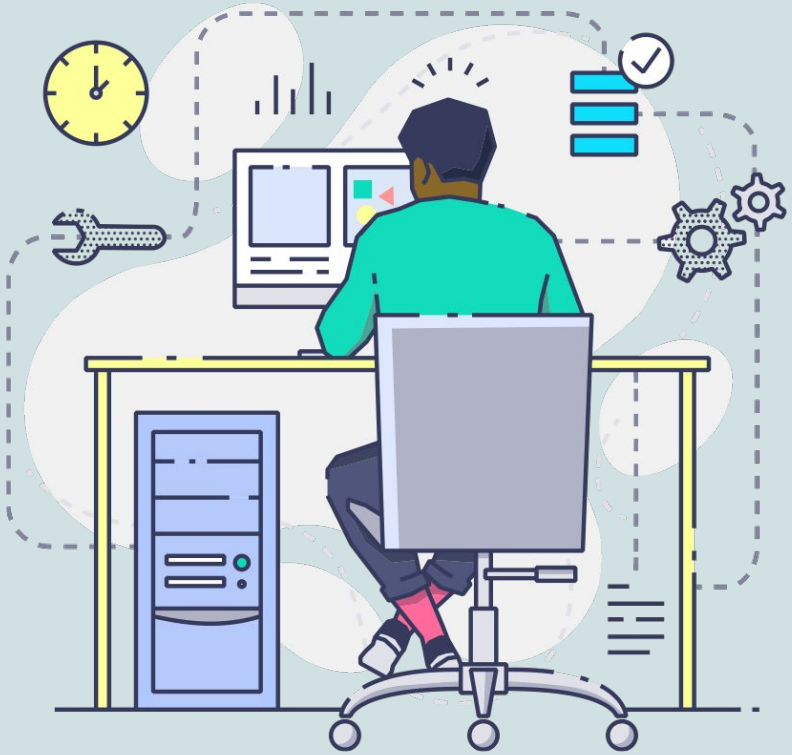
Example: Make a list of all users with old passwords, and force these users to update their credentials.

Write a script to automatically solve a problem.

Example: Write the script **`find_stale_users.sh`** to fix the issue.

Run the script and email the results regularly, using `cron`.

Example: Schedule **`find_stale_users.sh`** to run every Saturday at noon.



Overview of cron

Introducing `cron`

Consider the following scenario:

Before leaving work, you spend 10 minutes deleting your cache and your trash bin, and backing up your documents folder.

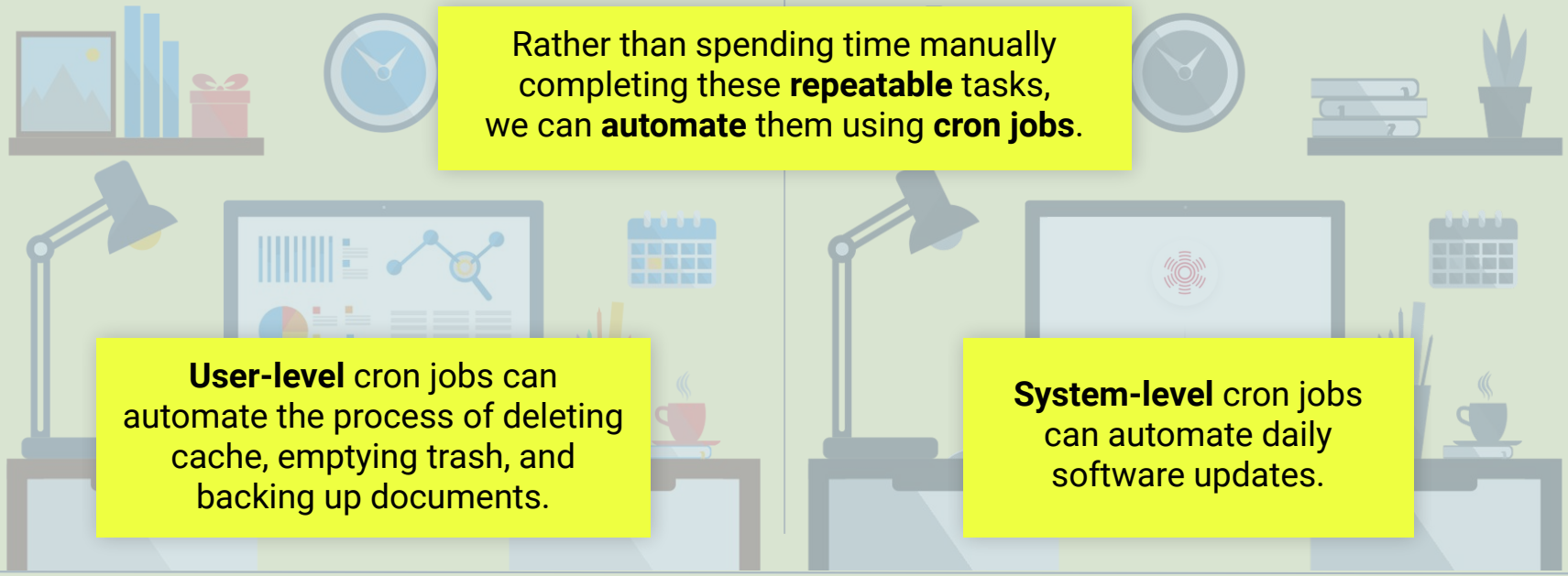
You also spend one hour per day installing software updates.

Introducing cron

Consider the following scenario:

Before leaving work, you spend 10 minutes deleting your cache and your trash bin, and backing up your documents folder.

You also spend one hour per day installing software updates.



Rather than spending time manually completing these **repeatable** tasks, we can **automate** them using **cron jobs**.

User-level cron jobs can automate the process of deleting cache, emptying trash, and backing up documents.

System-level cron jobs can automate daily software updates.

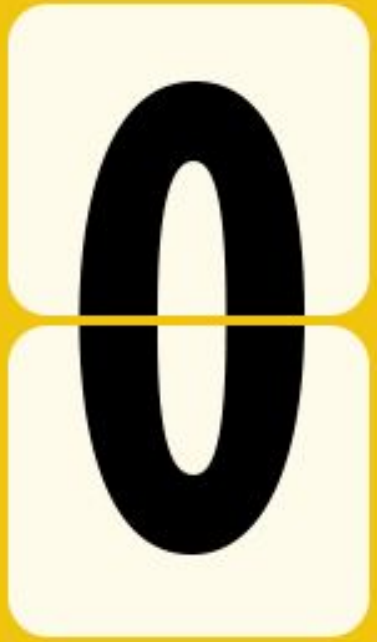
A **cron job** is a script or command designated to run at periodic, predetermined intervals.

Cron

A **daemon** is a computer program that runs as a background process, rather than being directly controlled by an interactive user.

`cron` is a robust task scheduler that allows users to schedule repetitive tasks to run on a regular basis.

Daemons are often started at boot up. While other daemons respond to network requests and hardware activity, `cron` is initiated at designated time intervals.



crontabs

Tasks are stored and scheduled in a file called a **crontab** (*cron table*).

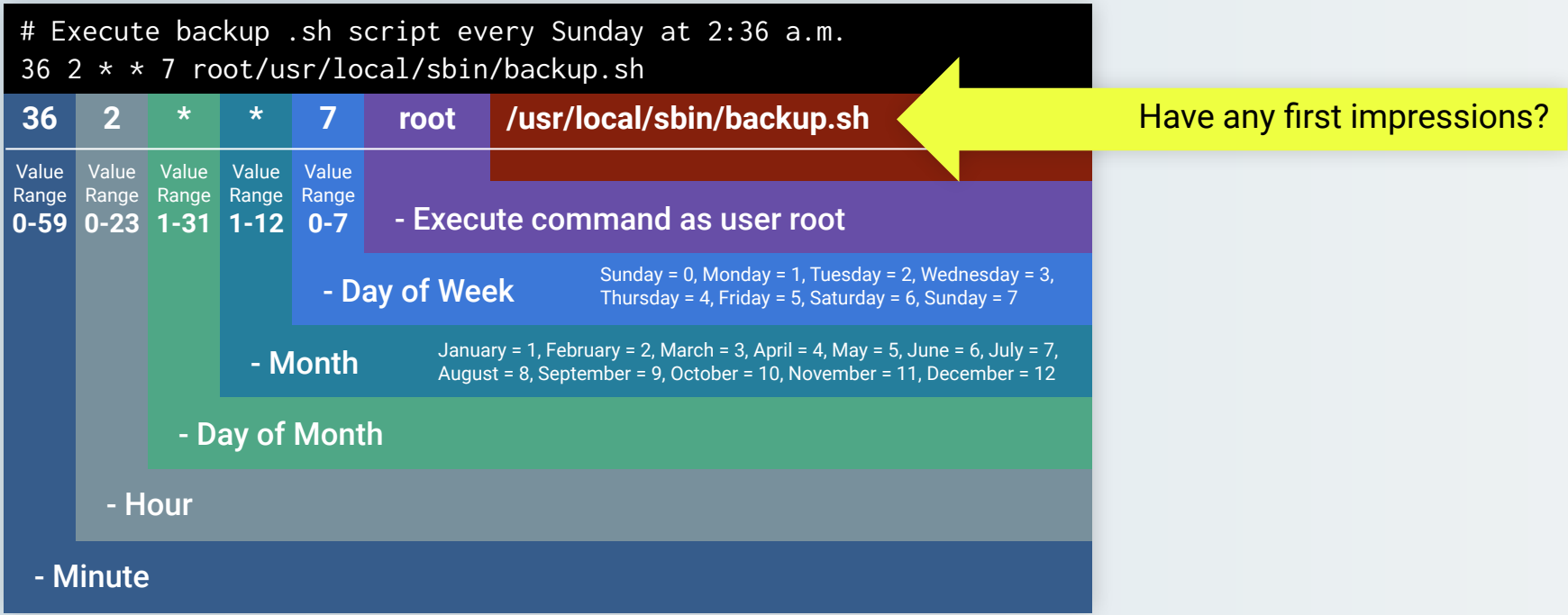
- Each user has their own individual, private crontab.
- Tasks scheduled by users are run under that user's privilege level.
- Root users can have a crontab too.

(*Do you think this can pose threats?*)

cron Syntax

While they may seem intimidating, crontab rules can be learned relatively easily with some practice.

Let's look at crontab on the command line.

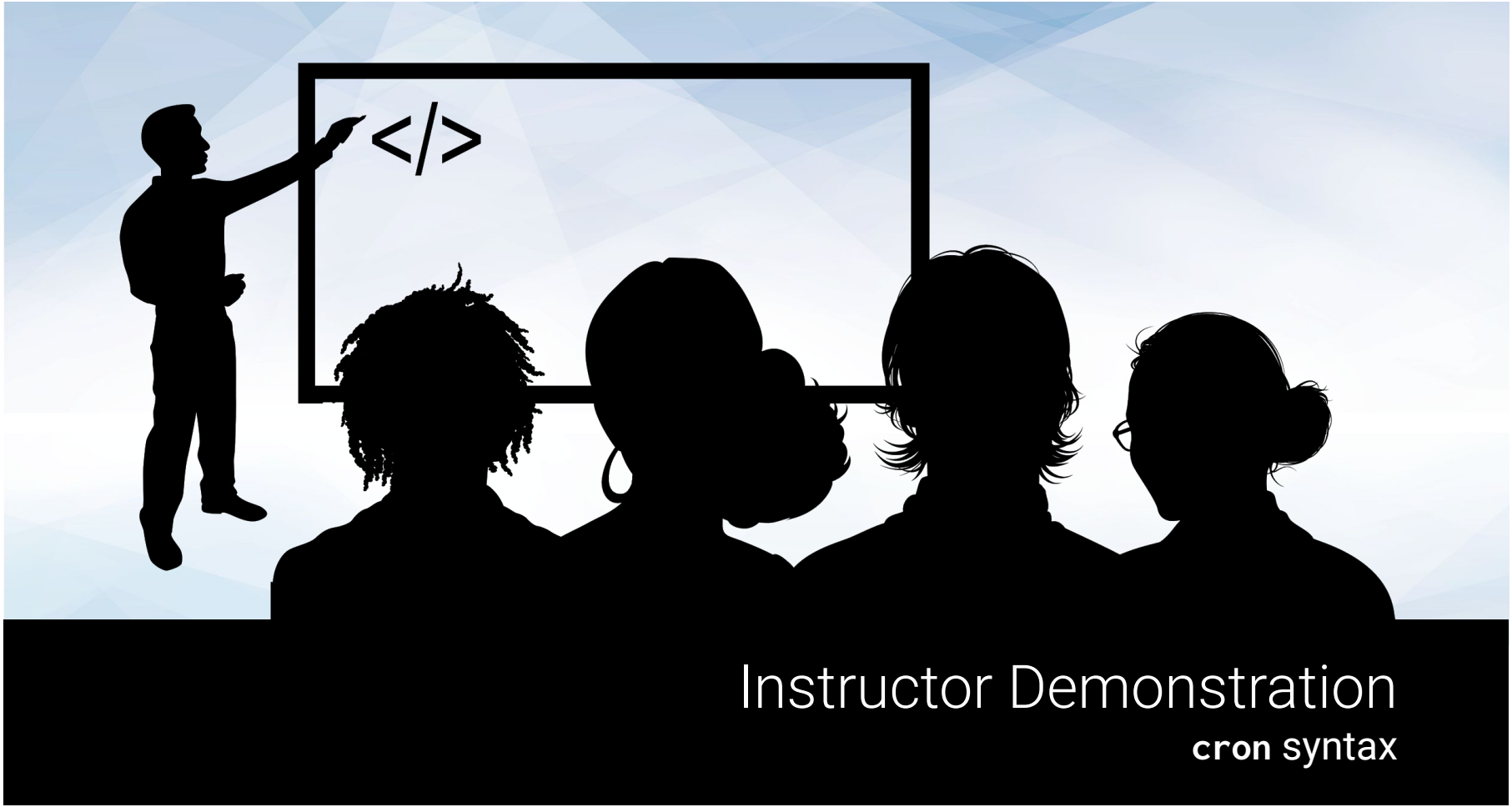


Source: linuxconfig.org

`cron` Syntax Walkthrough

In the upcoming demo, we will focus on the following:

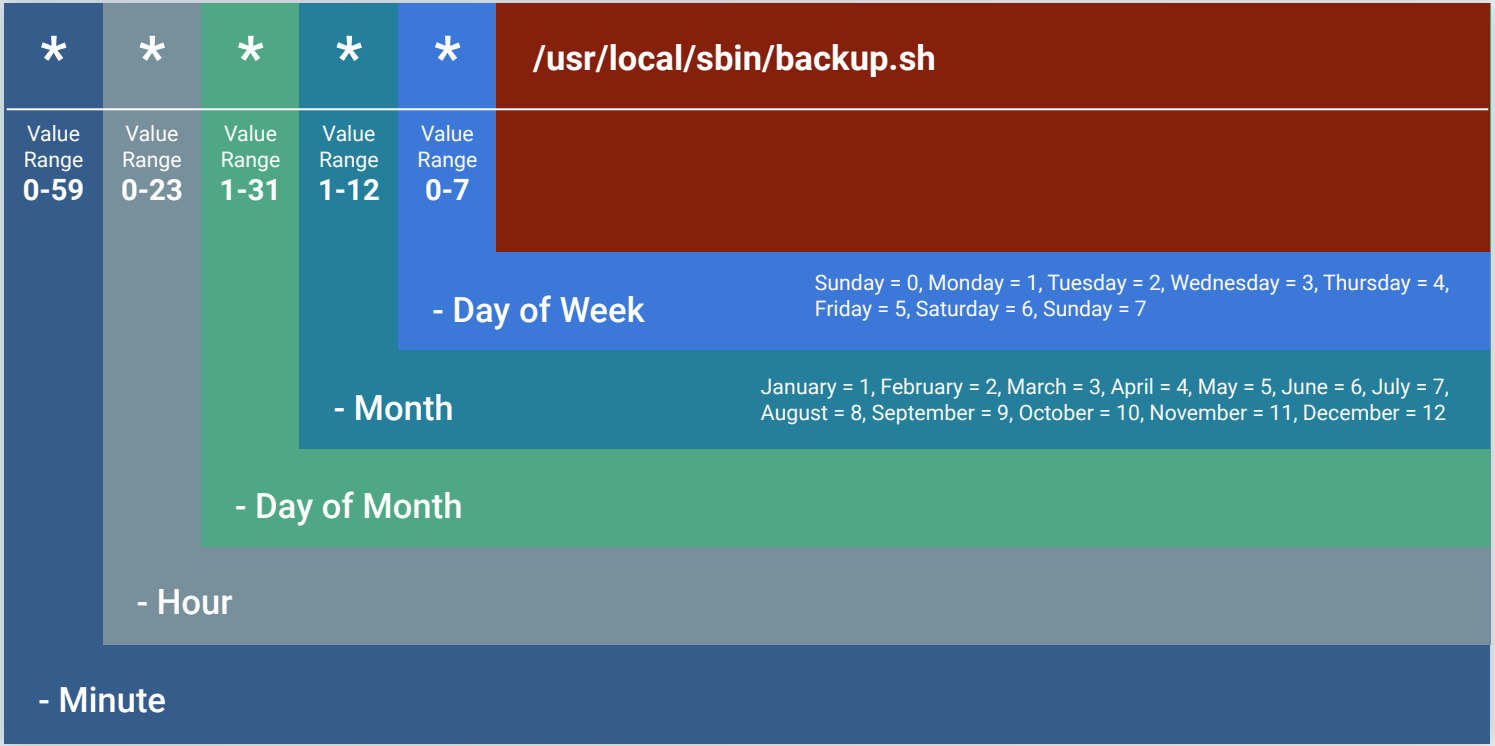
1. General `cron` command line syntax.
2. Editing a crontab with `crontab -e`.
3. Listing the contents of the crontab using `crontab -l`.



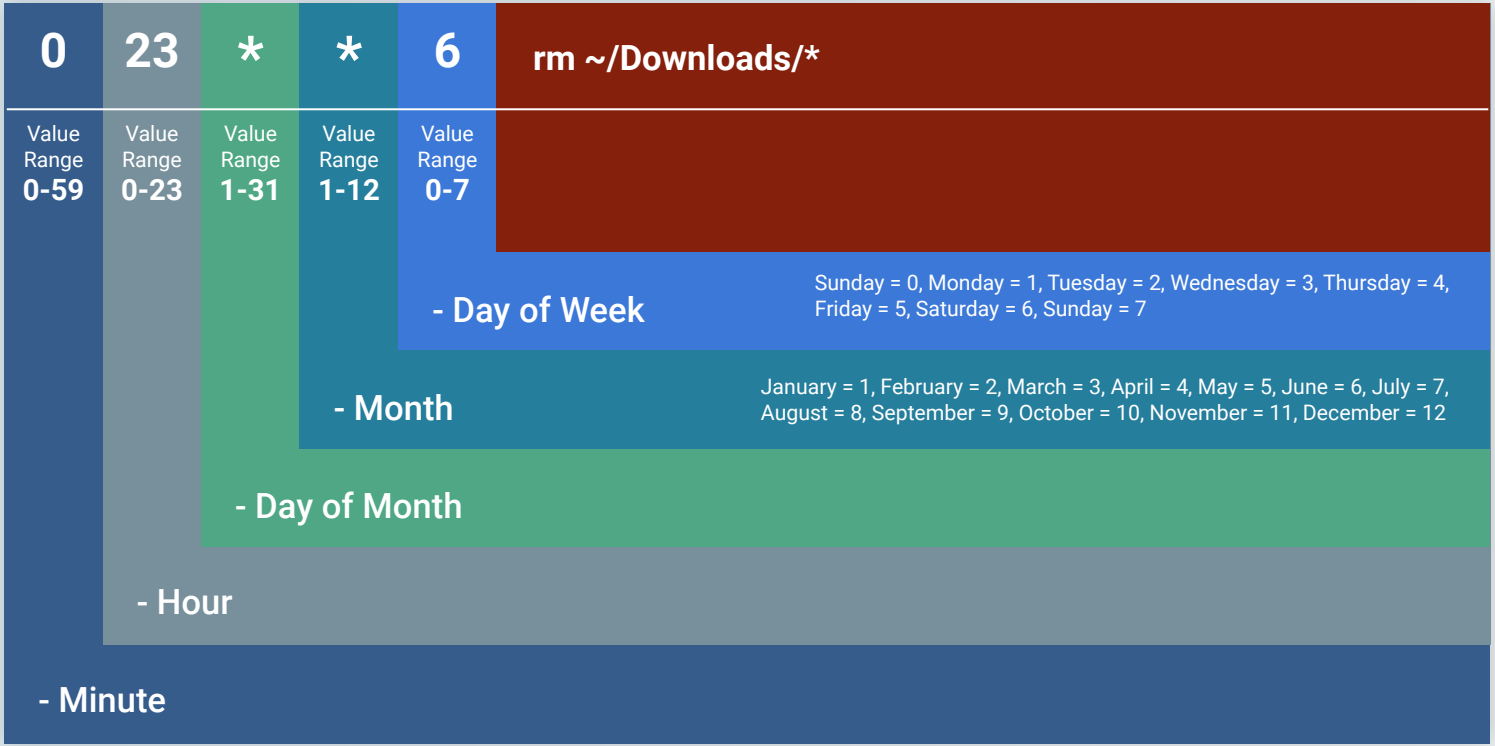
Instructor Demonstration

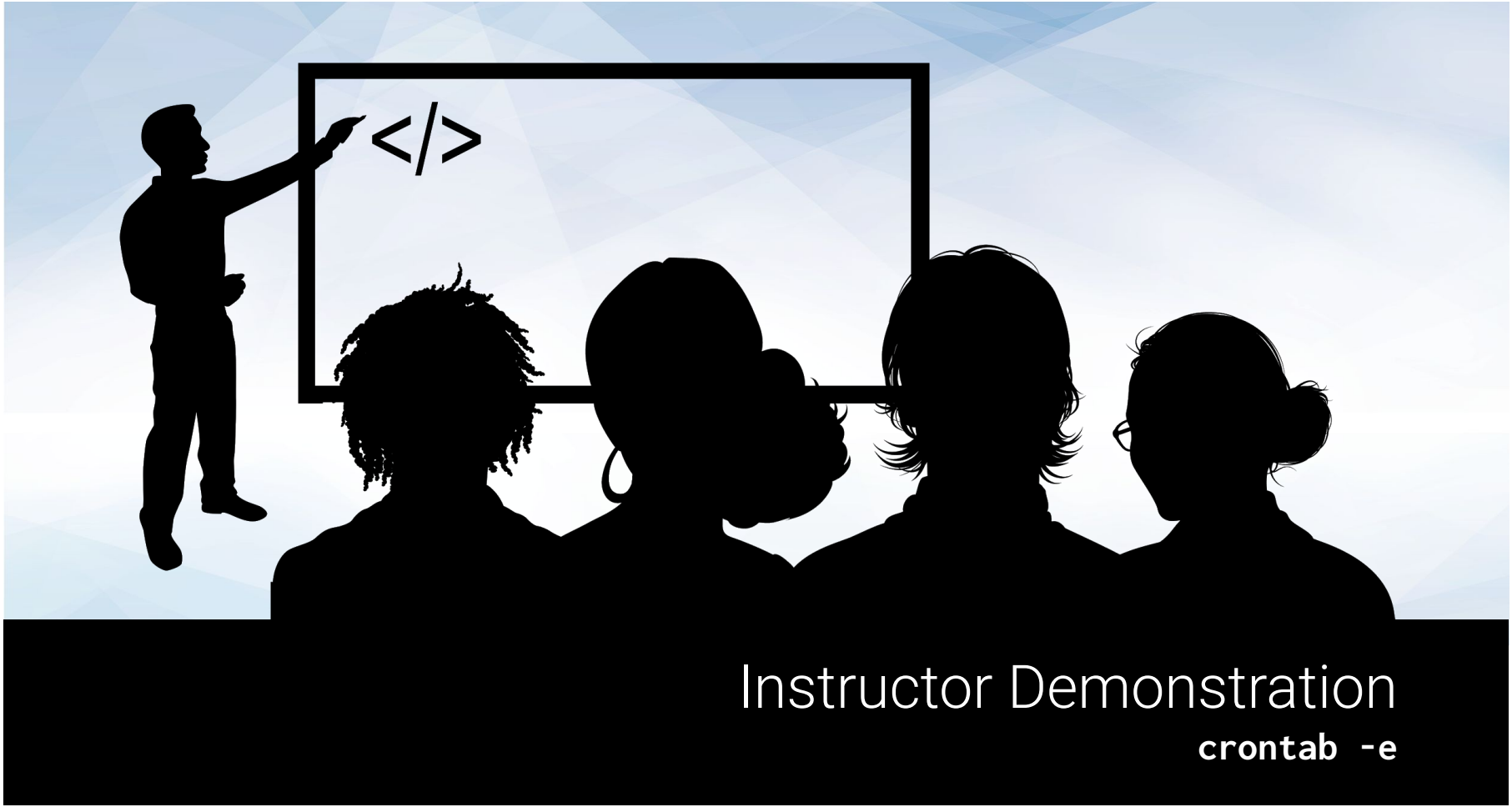
cron syntax

cron jobs


* * * * *
/usr/local/sbin/backup.sh
Value Range 0-59
Value Range 0-23
Value Range 1-31
Value Range 1-12
Value Range 0-7
- Day of Week
Sunday = 0, Monday = 1, Tuesday = 2, Wednesday = 3, Thursday = 4, Friday = 5, Saturday = 6, Sunday = 7
- Month
January = 1, February = 2, March = 3, April = 4, May = 5, June = 6, July = 7, August = 8, September = 9, October = 10, November = 11, December = 12
- Day of Month
- Hour
- Minute

cron jobs


0
23
*
*
6
rm ~/Downloads/*
Value Range 0-59
Value Range 0-23
Value Range 1-31
Value Range 1-12
Value Range 0-7
- Day of Week
Sunday = 0, Monday = 1, Tuesday = 2, Wednesday = 3, Thursday = 4, Friday = 5, Saturday = 6, Sunday = 7
- Month
January = 1, February = 2, March = 3, April = 4, May = 5, June = 6, July = 7, August = 8, September = 9, October = 10, November = 11, December = 12
- Day of Month
- Hour
- Minute



Instructor Demonstration

`crontab -e`

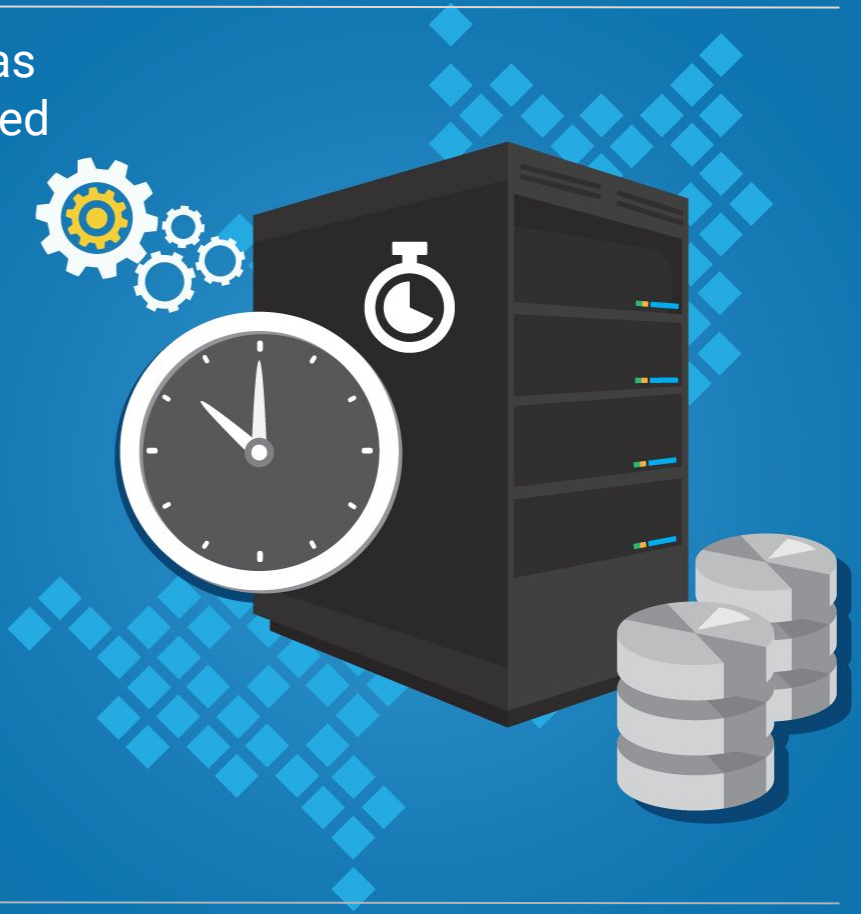
crontab

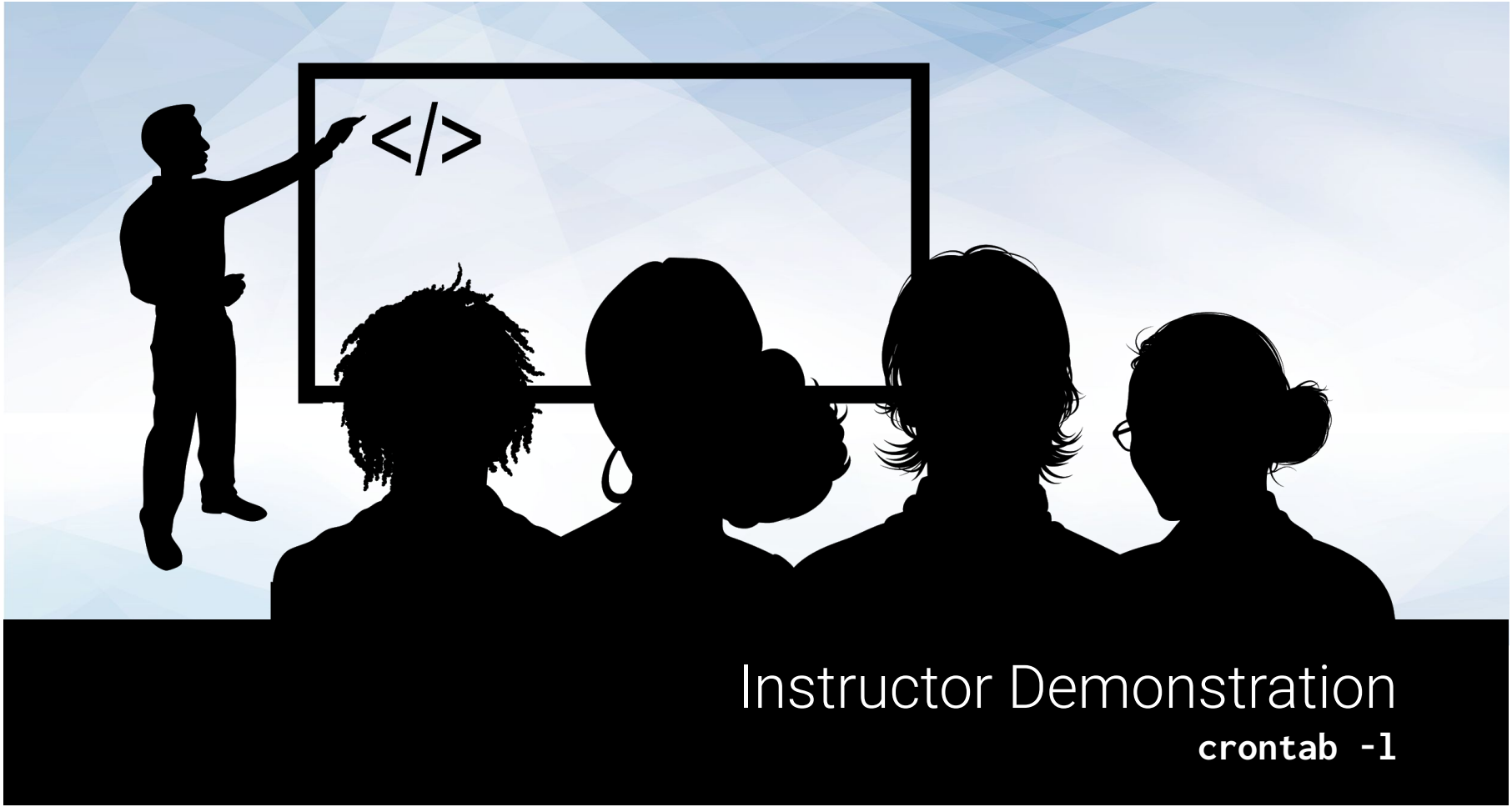
Cron jobs run under the same permissions as the user who creates them. A cron job created by user `root` will run with root privileges.

This introduces security risks. Anyone capable of privilege escalation can add a malicious cron job to the root crontab.

- We'll revisit the risks of using `cron` with root privileges later in the unit.
- Best practice is to avoid using the root crontab.

Inspecting the root crontab for unauthorized or malicious entries is a critical step in ensuring the integrity of any system that uses it. **Let's take a look.**





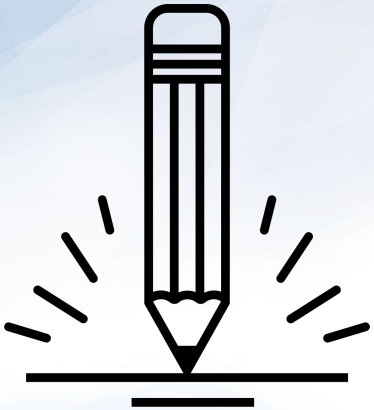
Instructor Demonstration

`crontab -l`

Today's Scenario

In today's activities, we'll act the role of a junior administrator at the company Rezifp Pharma Inc.

- There has been a wave of recent ransomware attacks. You will be responsible for using `cron` to automate backup jobs that contain files in the E-Prescription Treatment database.
- Rezifp maintains a large number of files related to patients, doctors, and treatments.
- Administrators at various clinics often create files that contain Personal Identifiable Information or (PII) such as email addresses, passwords, biometric records, etc.



Activity: Simple Cron Jobs

In this activity, you will play the role of a junior administrator at a pharmaceutical company, tasked with using `cron` to automate the backup of treatment databases.

Activity file shared by the instructor.

Suggested Time:
15 Minutes

Time’s Up! Let’s Review.

Activity Review: Simple Cron Jobs

Completing this activity required the following steps:

01 Using `systemctl` to verify that the `cron` daemon is installed and running.

02 Using `crontab -l` to inspect user crontabs and verify validity.

03 Using `crontab -e` to edit user crontab files.

04 Using crontab to automate `cron` jobs to move and archive files and directories.

05 Verifying archives *after they are written* to check for errors.

Let’s Review

In the previous section, we learned:

- Crontabs come in two varieties:
  - User-level: runs for a specific user under their privilege level.
  - System-level: runs for the system as a whole under root privileges.
- User-level crontabs are often used for “personal” tasks, such as organizing files.
- The general syntax for a crontab:

```
minute hour day-of-month month day-of-week command
```

- Tools like crontab.guru can verify the syntax of cron jobs before they run.

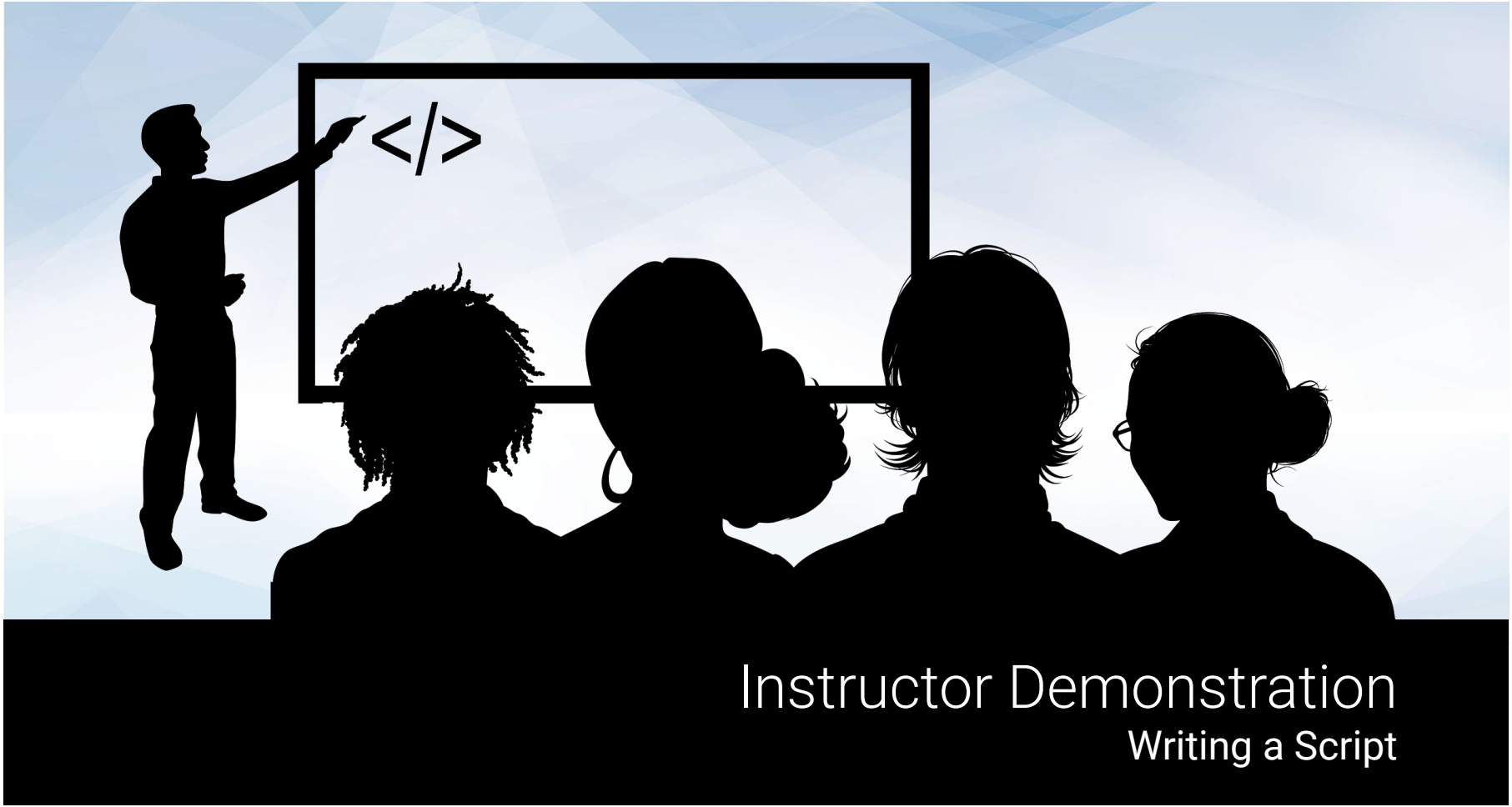
Introduction to Scripts

Why Scripts?

`cron` is useful for single tasks, like backing up a single user's directory, but not for backing up several users' directories.

- Scripts allow us to complete complex tasks, like creating backups or cleaning up multiple directories, by executing a single script.
- This results in cleaner crontabs with fewer lines of code, and allows us to schedule complex jobs that can't be expressed by a single command.

Next, we'll use scripting to schedule several functions to run at once, once per day.

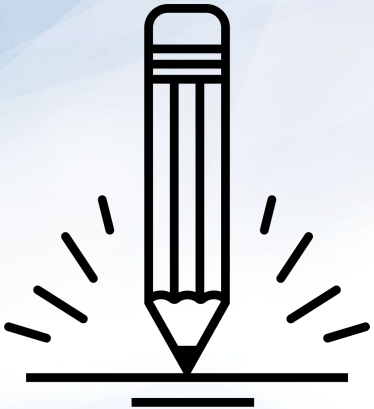


Instructor Demonstration

Writing a Script

Demo Summary

In the previous walkthrough, we did the following:



Activity: Scripting

In this activity, you will work the role of a junior administrator tasked with creating a shell script that keeps your system clean, up-to-date, and ensures backups remain current and uncorrupted.

Activity file shared by the instructor.

Suggested Time:
20 Minutes

Time’s Up! Let’s Review.

Activity Review: Scripting

Completing this activity required the following steps:

01 Creating a directory to hold your scripts in `~/Security_scripts`.

02 Writing shell script `backup.sh` to automate gzip-compressed archives and backups.

03 Writing shell script `update.sh` to automate software package updates and removal.

04 Writing shell script `cleanup.sh` to automate the cleanup of cached files and generate a report of system resource usage.

05 Testing the scripts by running them with bash using the `./name.sh` command.

Countdown timer

15:00

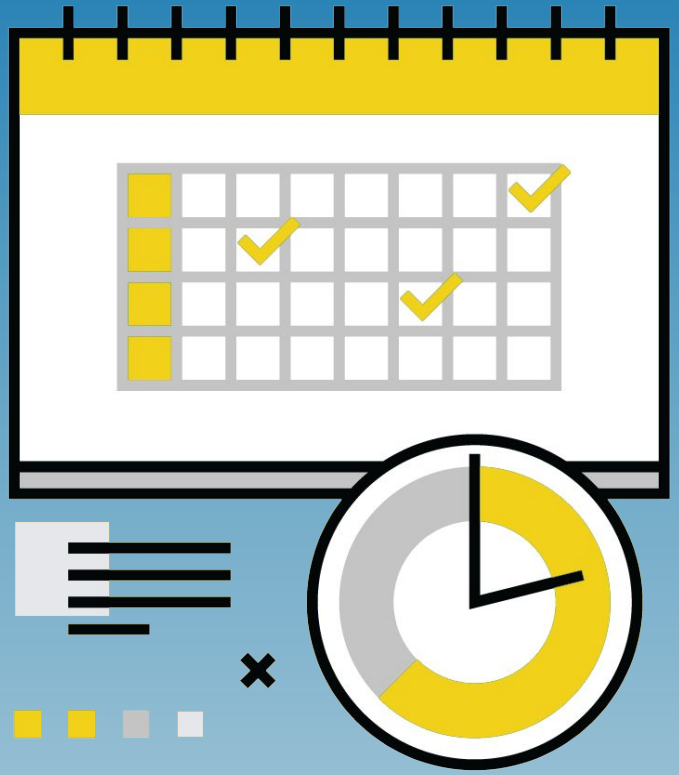
(with alarm)

Scheduling Backups, Cleanups, and Security Checks

Scheduling Jobs for Scripts

Remember: user cron jobs are not very useful for system-level maintenance.

- Cron jobs belong to individual users, so they are typically unable to modify the whole system without root privileges.
- Now, we will use **system-wide cron directories** to run cron jobs with root privileges.



System-Wide Cron Directories

System-wide cron directories are located at:

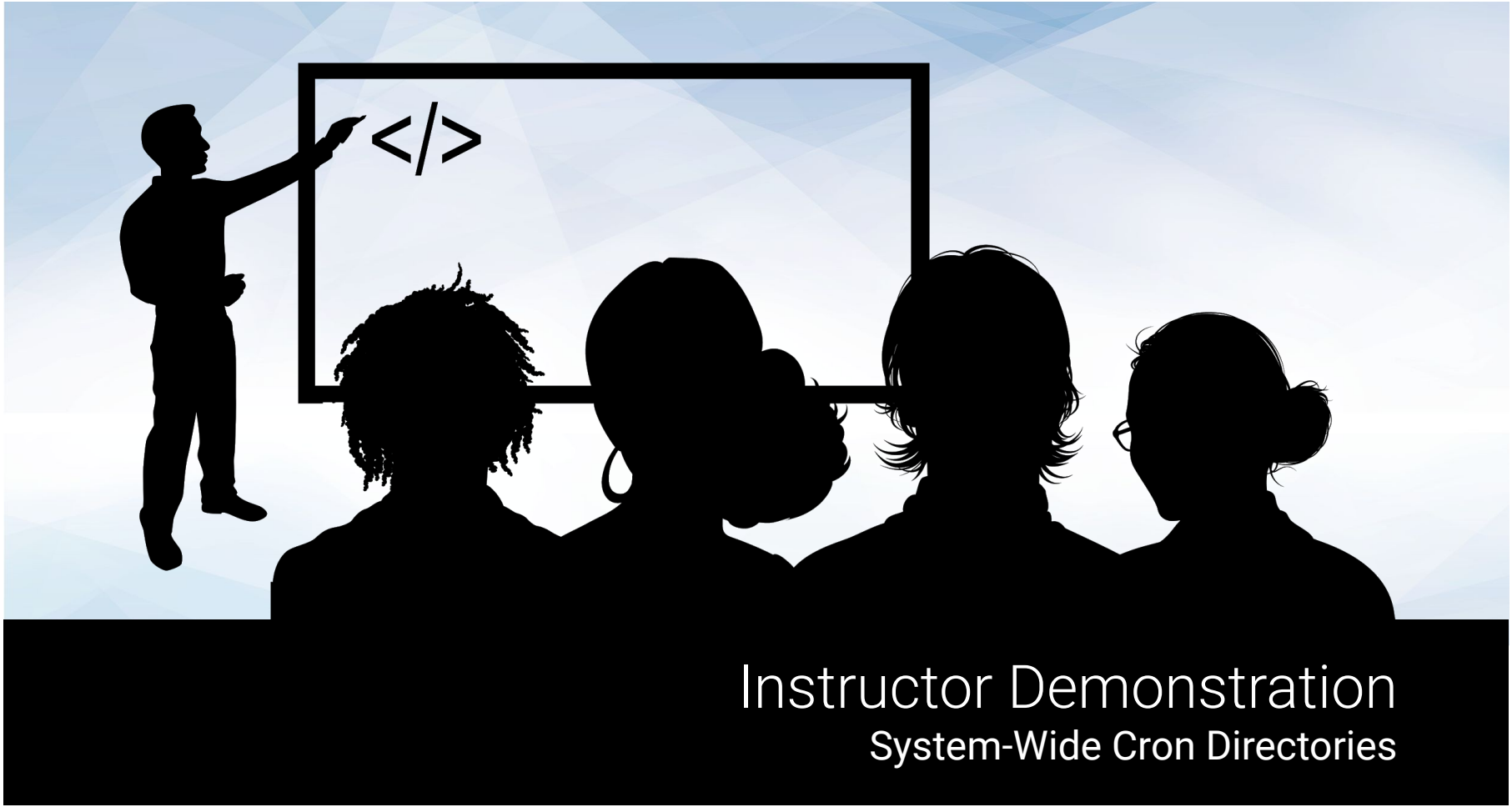
- `/etc/cron/d`
- `/etc/cron.daily`
- `/etc/cron.weekly`
- `/etc/cron.monthly`

Each directory contains several scripts that run at the dedicated time intervals.

- For example, scripts placed in `/etc/cron.weekly` will run once per week.

Next,
we'll take a look

into the `etc/cron` `daily`, `weekly`, and `monthly` directories.



Instructor Demonstration

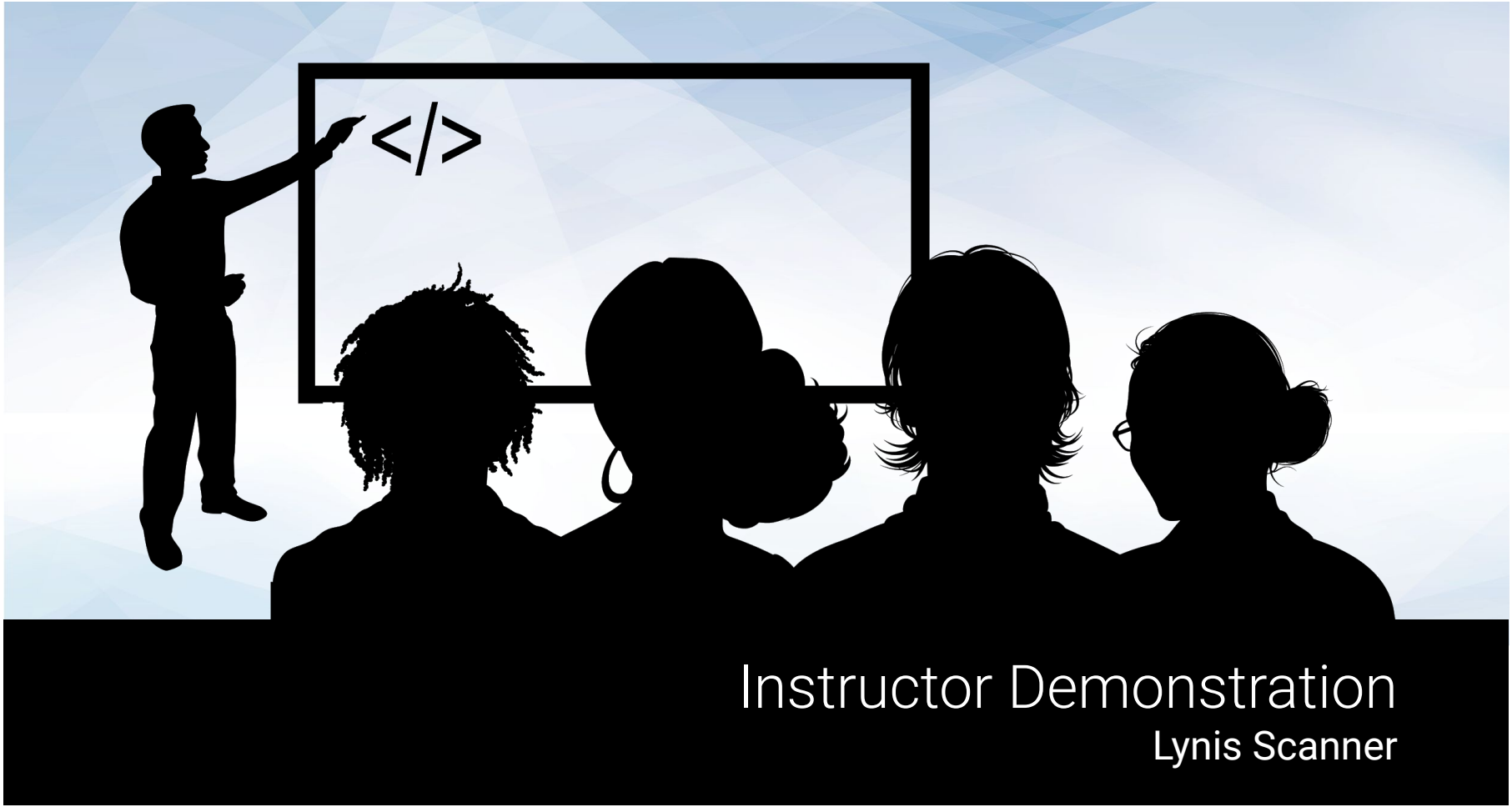
System-Wide Cron Directories

Lynis Scanner

Lynis is a security scanner used to check a machine for vulnerabilities.

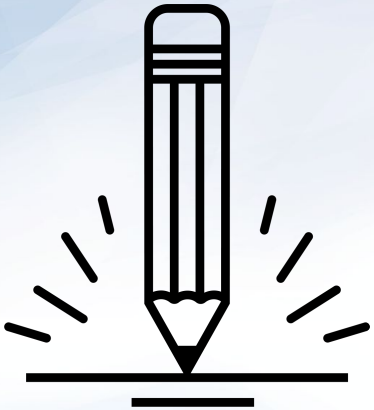
- Generates and saves reports of its findings for administrators to review.
- Offers numerous scan types.

Today, we'll experiment with a few different ones...



Instructor Demonstration

Lynis Scanner



Activity: Scheduling Backups and Cleanups

In this activity, you will work the role of a junior administrator tasked with creating creating system-wide cron directories to schedule your previously made scripts.

Activity file sent via Instructor.

Suggested Time:
15 Minutes

Time’s Up! Let’s Review.

Activity Review: Scheduling Backups and Cleanups

Completing this activity required the following steps:

01 Move the `backup.sh`, `cleanup.sh`, and `update.sh` scripts to their corresponding system-wide cron directories.

02 Create `lynis` scripts to perform security scans.

Attacking `cron`

Attackers will exploit job scheduling to execute programs at system startup or on a scheduled basis to establish **persistence**.

What is persistence?

The key to a profitable malware campaign is persistence.

Once malware is on a system, attackers want to *keep* it on the system. Malicious code that is easily detected and removed will not generate enough value for the attacker.

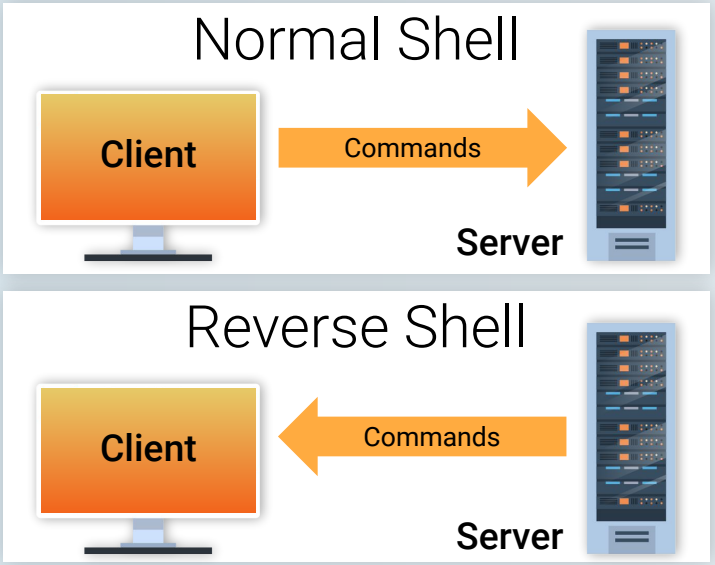
Identifying `cron` Exploits

Today, we'll learn how to address a specific **`cron` persistence mechanism** by analyzing network connections, terminating them, and scrubbing the system of malicious files.

The persistence mechanism we're looking at establishes **a reverse shell** to a remote host and exploits **`cron`** to maintain a connection to the victim's machine, even after reboot.

Reverse Shells

A reverse shell is a type of shell in which the victim machine communicates back to the attacking machine. The attacking machine has a listener port that receives the connection, where code or command execution is achieved.



Reverse shells allow attackers to:

- Gain full control of the system they are installed on.
- Collect and send your data out of your network.
- Capture username and passwords.
- Scan your network from the inside.

Reverse shells

Reverse shells can be installed on a host in a number of ways:

- Physical access.
- An insider installs a reverse shell.
- Someone is social engineered into installing the reverse shell program.
- A user executes email attachments that install the reverse shell program.
- A user downloads and executes reverse shell programs.
- A legitimate program acts like a reverse shell.

Netcat

One of the most popular tools used by attackers is called **Netcat** (**nc**).

- Netcat is a networking utility that uses TCP/IP protocol to read and write data across network connections.
- Netcat is designed to be a reliable “back-end” tool that can be used directly or easily driven by other programs and scripts to offer a backdoor for attacks.

Netcat Syntax Example

```
$ nc -n -v -l -p 1234
```

[command] [-option] [port]

- **-n** Do not do any DNS or service lookups on any specified addresses, hostnames or ports.
- **-v** Provide more verbose output.
- **-l** Listen for an incoming connection rather than initiate a connection to a remote host.
- **-p** Specifies the source port **nc** should use, subject to privilege restrictions and availability.

Netcat Output Example

```
$ nc -n -v -l -p 1234
listening on [any] 1234...

Connect to [192.168.1.2] from (UNKNOWN) [192.168.1.3] 36266
bash: cannot set terminal process group (5878): Inappropriate ioctl for device
bash: no job control in this shell
```

- **`listening`** on **`[any]`** is listening on any [IP] on port 1234
- **`Connect to`** is the IP address of the remote host or victim.
- **`from`** is the IP address of the attacker's machine.

Mitigating Reverse Shells

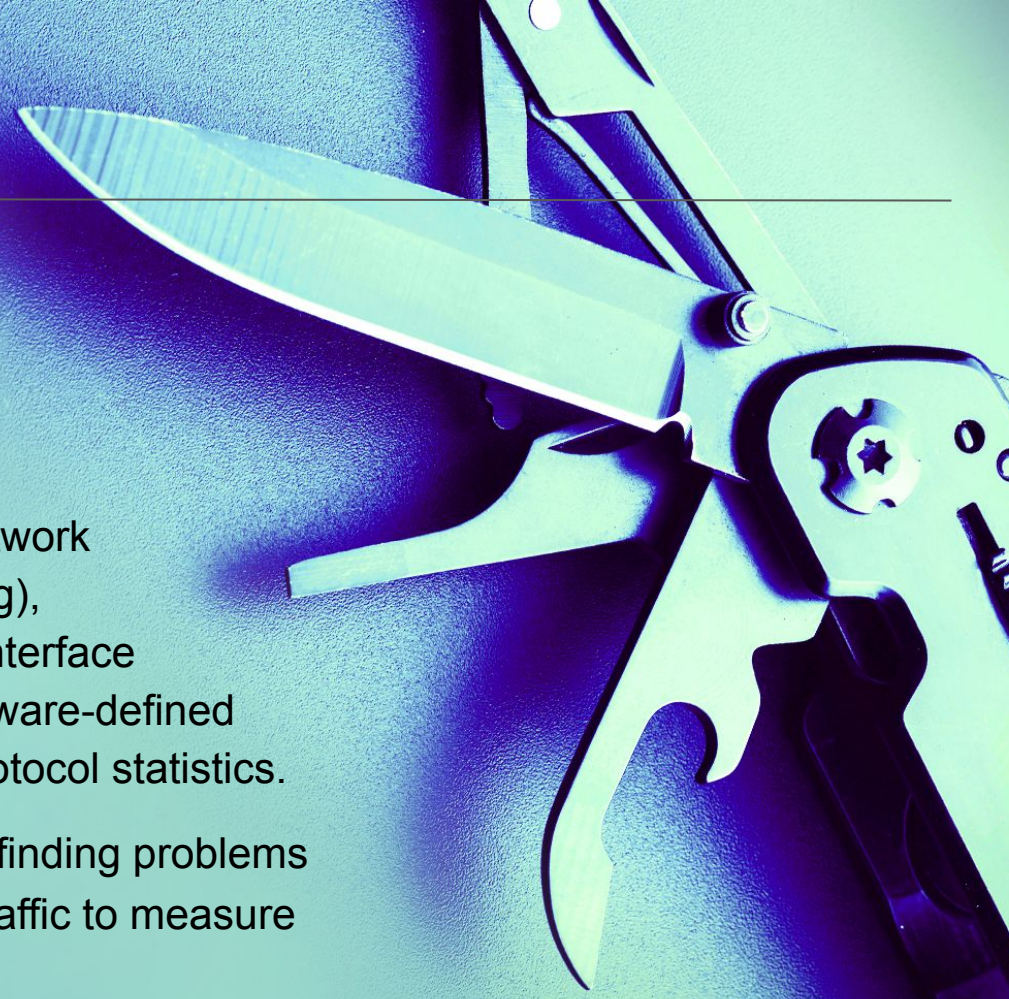
We can check for reverse shells on a computer by analyzing network connections.

Look for key indicators such as established TCP connections, foreign IP address and ports, process IDs, and program names.

Netstat

We'll use the **`netstat`** command to display network connections that indicate a reverse shell.

- **`netstat`** (*network statistics*) is a command-line tool that displays network connections (incoming and outgoing), routing tables, and many network interface (network interface controller or software-defined network interfaces) and network protocol statistics.
- **`netstat`** is a very versatile tool for finding problems in the network and using network traffic to measure performance.



Netstat Syntax Example

```
$ netstat -a -p -t
```

[command] [-option]

-a Displays all active connections and the TCP and UDP ports on which the computer is listening.

-t Displays only TCP connections.

-p Displays which processes are using which sockets. (You must be `root` to do this).

The `kill` Command

There are several ways to terminate processes. Today we'll use the **`kill`** command.

```
$ kill 1234
```

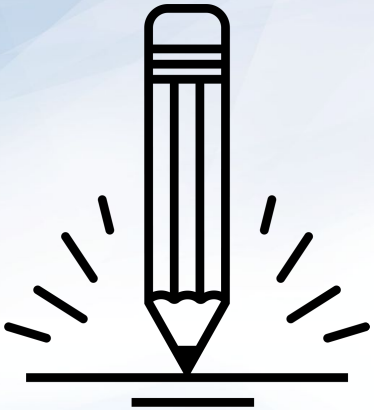
`[command] [PID]`

- A TCP connection can be killed by either its program name or PID (preferred).
- Pay special attention to killing PIDs as they can be a shared process between two separate operations or programs.

Detecting Reverse Shells

There are several ways to mitigate reverse shells that leverage `cron`:

- Scrutinize drop logs on firewall and proxies.
- Tune IDS to alert on unexpected traffic.
- Review the logs of an email server that starts browsing the web.
- Review the logs of DNS servers that telnet out of network.
- Review host-based firewall logs.
- Check server baselines against known good configurations.
- Install application-aware firewalls and proxies.



Activity: Attacking `cron`

In this activity, you will work as a junior administrator to enact security measures that ensure the integrity of the organization by identifying rogue connections, understanding how they were connected, and then killing the root causes.

Activity file shared by the instructor.

Suggested Time:
20 minutes

Time’s Up! Let’s Review.

Activity Review: Attacking `cron`

Completing this activity required the following steps:

01 Use `sudo su` to acquire root privilege user access.

02 Use `netstat -atp` to check for established TCP connection.

03 Use `sudo crontab -l` to inspect the root user's crontab.

04 Use `nc -nvlp 888` to open a listener in an attacker's terminal.

05 Use `kill` (PID) to terminate an active TCP connection.

06 Use `crontab -e` to edit the root user's crontab.

Any Questions?