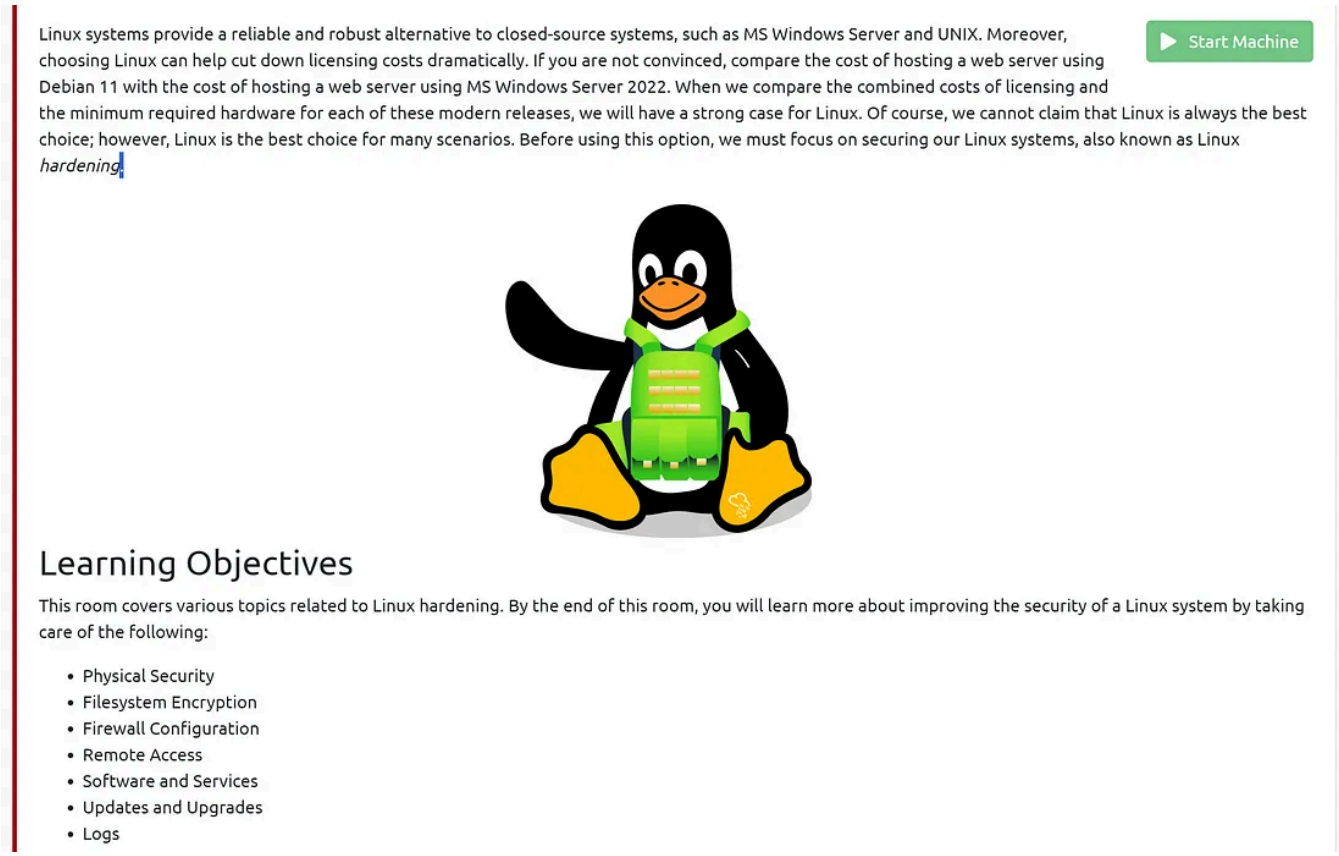
Tryhackme: Linux System Hardening

Daniel Schwarzentraub · Follow
5 min read · Sep 28, 2023

Task 1: Introduction



Prerequisites

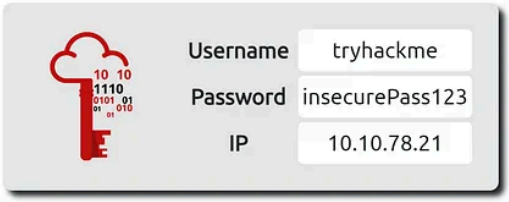
We recommend that users have a good working knowledge of the Linux OS and a solid understanding of the basic security principles. We recommend the Linux Fundamentals module if you want to learn about Linux. If you haven't done so, we suggest you complete the Security Principles room.

Linux VM

Some of the tasks in this room require using the attached Linux VM. We suggest that you click on the Start Machine button, along with the Start AttackBox button, so that they are ready when you need them. You may also use your VPN connection to connect with the Linux VM. You can access the attached machine using `ssh` by supplying the following credentials:

- Username: `tryhackme` (i.e., `ssh tryhackme@10.10.78.21`)
- Password: `insecurePass123`

It is worth repeating that `insecurePass123` is a weak password; however, a complicated password such as `HkAcPQqdXKC2` would be very cumbersome to type while going through the exercises in this room. But please remember that a weak yet convenient password would put the security of your whole system and network at risk.



Task 2: **Physical Security**

One of the security principles is Defence-in-Depth. Hence, we should always think in terms of layers of security. One of the first layers is physical security.

It would be best if you prevented potential adversaries from being able to gain physical access to your computer systems. If an intruder can gain physical access to your office, it would be easy to remove the disk drive and take it away. That's a simple attack that requires minimal technical skills.

Let's say you have taken the necessary measures to prevent intruders from taking the disk drive or the whole computer system. Moreover, you have ensured that your system passwords are complex and impossible to guess. If an intruder can access the system physically, it is a non-sophisticated task to use GRUB, a popular Linux bootloader, to reset the root password account. Hence we have the adage "boot access = root access".

It is evident that we need to ensure physical security for our computer systems; however, in the unlikely event that physical security is breached, we need to provide additional layers of protection. Many BIOS and UEFI firmware allows you to add a boot password. This password will prevent unauthorised users from booting the system. However, this can only be used for personal systems; it won't make sense to use it on servers as this will require someone to be physically present to supply the boot password.

We can consider adding a GRUB password depending on the Linux system we want to protect. Many tools help achieve that. One tool is `grub2-mkpasswd-pbkdf2`, which prompts you to input your password twice and generates a hash for you. The resulting hash should be added to the appropriate configuration file depending on the Linux distribution (examples: Fedora and Ubuntu). This configuration would prevent unauthorised users from resetting your root password. It will require the user to supply a password to access advanced boot configurations via GRUB, including logging in with root access.

AttackBox Terminal

```
root@AttackBox# grub2-mkpasswd-pbkdf2
Enter password:
Reenter password:
PBKDF2 hash of your password is
grub.pbkdf2.sha512.10000.534B77859C13DCF094E90B926E26C586F5DC9D00687853487C4BB1500D57EC29E2D6D07A586262E093DCBDFF4B3552742A25700BA
```

It is important to note that adding a password for GRUB is not available for systems deployed using cloud service providers (such as our Linux VM); a GRUB password does not make sense as you don't have access to the physical terminal.

Ensuring proper physical security is a must considering how easy it is for an attacker with physical access to wreak havoc. Adding a password to GRUB is a reasonable measure to block users with physical access to a system's keyboard from gaining access. However, we need a plan in case an attacker finds a way to steal the disk drives.

What command can you use to create a password for the GRUB bootloader?

Answer: **grub2-mkpasswd-pbkdf2**

What does PBKDF2 stand for?

Answer: **Password-Based Key Derivation Function 2**

Task 3: **Filesystem Partitioning and Encryption**

Encryption makes data unreadable without the decryption key. In the scenario where an adversary has complete physical access to your laptop, for instance, by stealing it, we want to ensure that it won't be of any use to them. A disk drive full of encrypted data should be as good as a damaged one.

There are various software systems and tools that provide encryption to Linux systems. Since many modern Linux distributions ship with LUKS (Linux Unified Key Setup), let's cover it in more detail.

When a partition is encrypted with LUKS, the disk layout would look as shown in the figure below.

We have the following fields:

- **LUKS phdr**: It stands for *LUKS Partition Header*. LUKS phdr stores information about the UUID (Universally Unique Identifier), the used cipher, the cipher mode, the key length, and the checksum of the master key.
- **KM**: KM stands for *Key Material*, where we have KM1, KM2, …, KM8. Each key material section is associated with a key slot, which can be indicated as active in the LUKS phdr. When the key slot is active, the associated key material section contains a copy of the master key encrypted with a user's password. In other words, we might have the master key encrypted with the first user's password and saved in KM1, encrypted with the second user's password and saved in KM2, and so on.
- **Bulk Data**: This refers to the data encrypted by the master key. The master key is saved and encrypted by the user's password in a key material section.

LUKS reuses existing block encryption implementations. The pseudocode to encrypt data uses the following syntax:

```
enc_data = encrypt(cipher_name, cipher_mode, key, original, original_length)
```

As we can see, LUKS works with different ciphers and cipher modes. *Original* refers to the plaintext data of length, *original_length*. The user-supplied password is used to derive the encryption key; the key is derived using password-based key derive function 2 (PBKDF2).

```
key = PBKDF2(password, salt, iteration_count, derived_key_length)
```

Using a *salt* with a hash function repeating an *iteration count* ensures that the resulting key is secure enough for encryption. For more information, you might want to refer to the Introduction to Cryptography room.

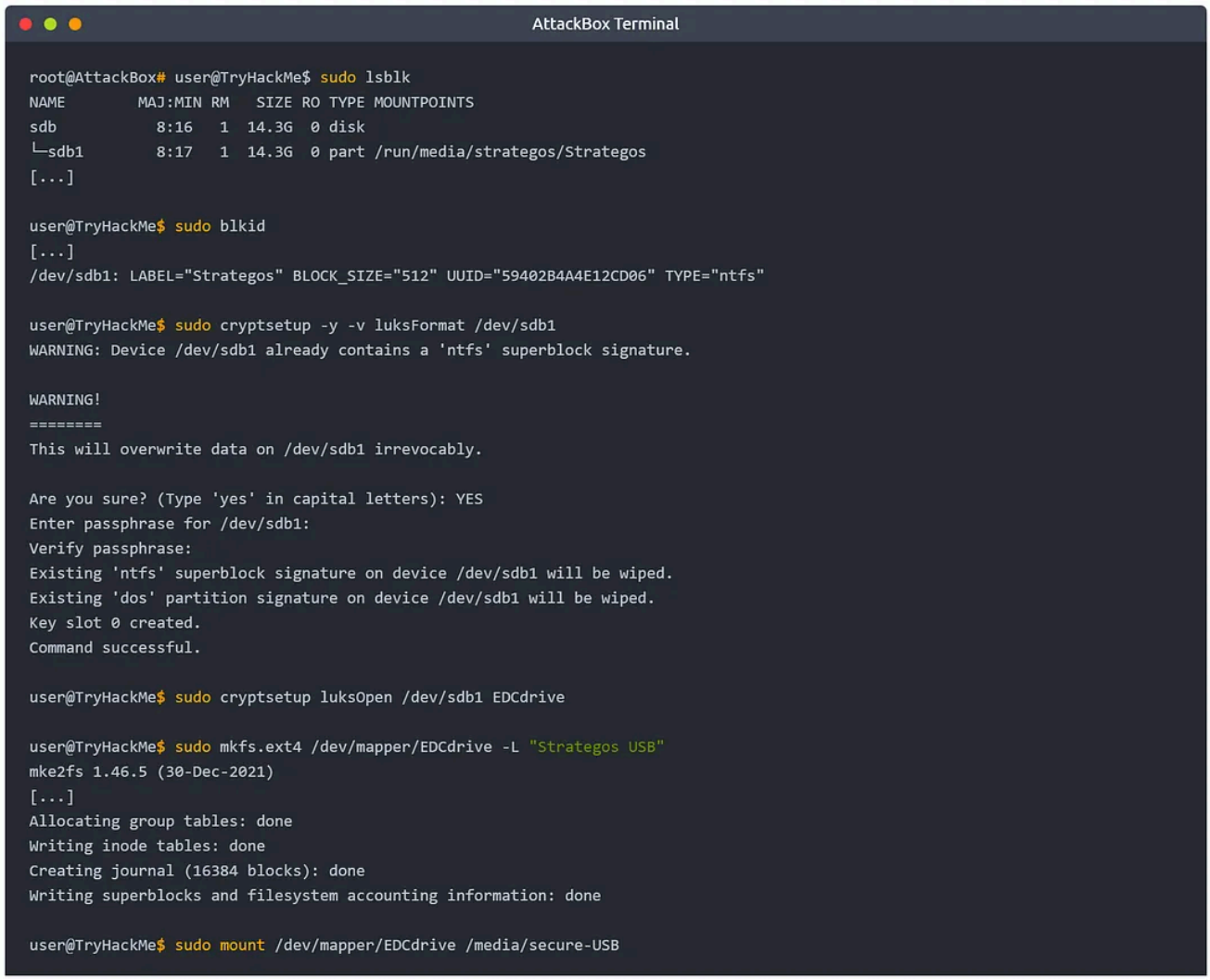
Similarly, to decrypt data and restore the original plaintext, LUKS uses the following syntax:

```
original = decrypt(cipher_name, cipher_mode, key, enc_data, original_length)
```

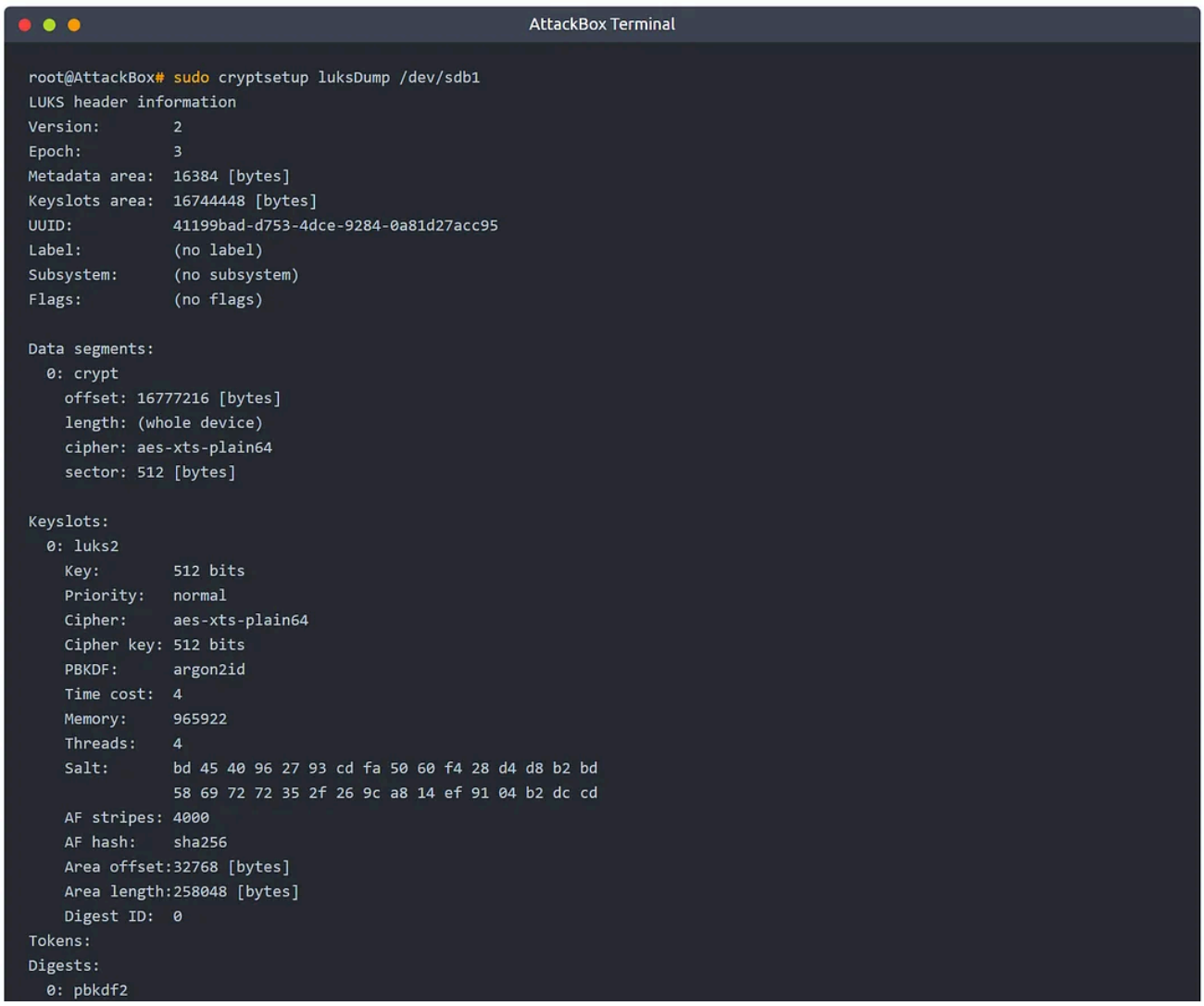
Most distributions let you encrypt a drive using a graphical interface. However, if you would like to set up LUKS from the command line, the steps are along these lines:

- Install `cryptsetup-luks`. (You can issue `apt install cryptsetup`, `yum install cryptsetup-luks` or `dnf install cryptsetup-luks` for Ubuntu/Debian, RHEL/Cent OS, and Fedora, respectively.)
- Confirm the partition name using `fdisk -l`, `lsblk` or `blkid`. (Create a partition using `fdisk` if necessary.)
- Set up the partition for LUKS encryption: `cryptsetup -y -v luksFormat /dev/sdb1`. (Replace `/dev/sdb1` with the partition name you want to encrypt.)
- Create a mapping to access the partition: `cryptsetup luksOpen /dev/sdb1 EDCdrive`.
- Confirm mapping details: `ls -l /dev/mapper/EDCdrive` and `cryptsetup -v status EDCdrive`.
- Overwrite existing data with zero: `dd if=/dev/zero of=/dev/mapper/EDCdrive`.
- Format the partition: `mkfs.ext4 /dev/mapper/EDCdrive -L "Strategos USB"`.
- Mount it and start using it like a usual partition: `mount /dev/mapper/EDCdrive /media/secure-USB`.

In the terminal below, we show a real example of encrypting a USB flash memory that initially has one partition at `/dev/sdb1` in NTFS format.


AttackBox Terminal
root@AttackBox# user@TryHackMe$ sudo lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
sdb 8:16 1 14.3G 0 disk
└─sdb1 8:17 1 14.3G 0 part /run/media/strategos/Strategos
[...]
user@TryHackMe$ sudo blkid
[...]
/dev/sdb1: LABEL="Strategos" BLOCK_SIZE="512" UUID="59402B4A4E12CD06" TYPE="ntfs"
user@TryHackMe$ sudo cryptsetup -y -v luksFormat /dev/sdb1
WARNING: Device /dev/sdb1 already contains a 'ntfs' superblock signature.
WARNING!
========
This will overwrite data on /dev/sdb1 irrevocably.
Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/sdb1:
Verify passphrase:
Existing 'ntfs' superblock signature on device /dev/sdb1 will be wiped.
Existing 'dos' partition signature on device /dev/sdb1 will be wiped.
Key slot 0 created.
Command successful.
user@TryHackMe$ sudo cryptsetup luksOpen /dev/sdb1 EDCdrive
user@TryHackMe$ sudo mkfs.ext4 /dev/mapper/EDCdrive -L "Strategos USB"
mke2fs 1.46.5 (30-Dec-2021)
[...]
Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
user@TryHackMe$ sudo mount /dev/mapper/EDCdrive /media/secure-USB

If you want to check the LUKS setting, you can issue the command `cryptsetup luksDump /dev/sdb1`. In the terminal output below, we can see the UUID of the encrypted disk. We can also see that the cipher used is `aes-xts-plain64`. As for the key, PBKDF2 used SHA256 with the provided salt for 194180 iterations.

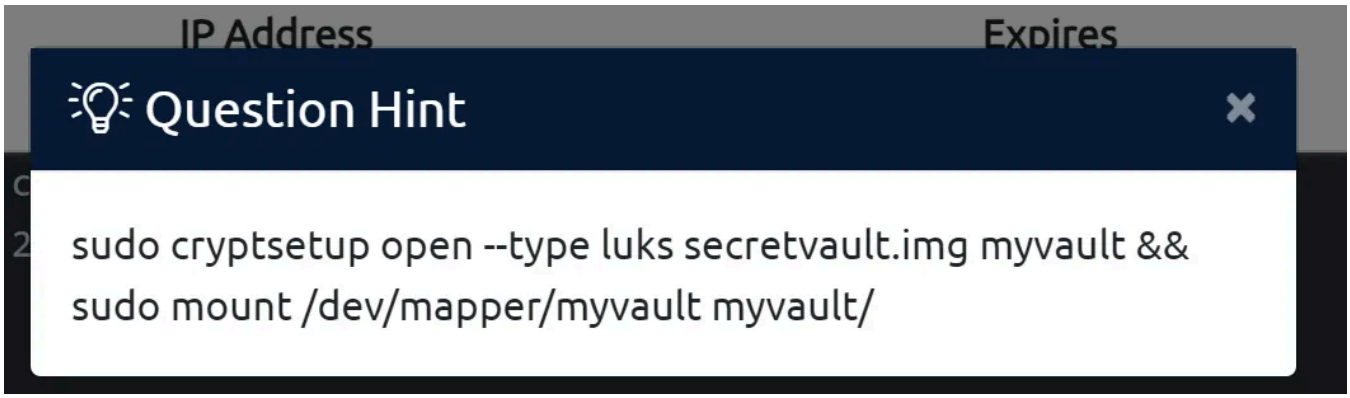


What does LUKS stand for?

Answer: **Linux Unified Key Setup**

We cannot attach external storage to the VM, so we have created a `/home/tryhackme/secretvault.img` file instead. It is encrypted with the password `2N9EdZYNkszEE3Ad`. To access it, you need to open it using `cryptsetup` and then mount it to an empty directory, such as `myvault`. What is the flag in the secret vault?

Let's take a look at the question hint




```
tryhackme@ip-10-10-78-21:~$ sudo cryptsetup open --type luks secretvault.img myvault && sudo mount /dev/mapper/myvault myvault/
Enter passphrase for secretvault.img:
tryhackme@ip-10-10-78-21:~$
```

```
tryhackme@ip-10-10-78-21:~$ cd myvault
tryhackme@ip-10-10-78-21:~/myvault$ ls
lost+found  task3_flag.txt
tryhackme@ip-10-10-78-21:~/myvault$ cat task3_flag.txt
THM{LUKS_not_LUX}
tryhackme@ip-10-10-78-21:~/myvault$
```

Answer: **THM{LUKS_not_LUX}**

Task 4: **Firewall**

> Let's briefly revisit the client/server model before we start. Any networked device can be a client, a server, or both simultaneously. The server offers a service, and the client connects to the server to use it. Examples of servers include web servers (HTTP and HTTPS), mail servers (SMTP(S), POP3(S), and IMAP(S)), name servers (DNS), and SSH servers. A server listens on a known TCP or UDP port number awaiting incoming client connection requests. The client initiates the connection request to the listening server, and the server responds to it.
>
> A firewall decides which packets can enter a system and which packets can leave a system. For more information about firewalls, we recommend you check the Firewalls room. Without a firewall, a client can communicate with any server without restrictions; moreover, a client can function as a server and listen for incoming connections from other clients. In other words, if an attacker manages to exploit a vulnerability on a system without a firewall in place, the attacker could use the exploit to listen on a chosen port number on the victim's machine and connect to it without any restrictions.
>
> Setting up a firewall offers many security benefits. First and foremost, firewall rules provide fine control over which packets can leave your system and which packets can enter your system. Consequently, firewall rules help mitigate various security risks by controlling network traffic between devices. More importantly, firewall rules can be devised to ensure that no client can act as a server. In other words, an attacker cannot start a reachable listening port on a target machine; the exploit can start a listening port, but the firewall will prevent all incoming connection attempts.
>
> A host-based firewall is a piece of software installed on a system we want to protect. Unlike a network-based firewall, the host-based firewall restricts network packets to and from a single host. The firewall has two main functions:
>
> - What can enter? Allow or deny packets from entering a system.
> - What can leave? Allow or deny packets from leaving a system.
>
> Imposing rules on the packets entering and leaving a system will significantly improve our security posture. Let's investigate how we can achieve this on a Linux system.

Linux Firewalls

The first Linux firewall was a packet filtering firewall, i.e., a stateless firewall. A stateless firewall can inspect certain fields in the IP and TCP/UDP headers to decide upon a packet but does not maintain information about ongoing TCP connections. As a result, a packet can manipulate a few TCP flags to appear as if it is part of an ongoing connection and evade certain restrictions. Current Linux firewalls are stateful firewalls; they keep track of ongoing connections and restrict packets based on specific fields in the IP and TCP/UDP headers and based on whether the packet is part of an ongoing connection.

The IP header fields that find their way into the firewall rules are:

1. Source IP address
2. Destination IP address

The TCP/UDP header fields that are of primary concern for firewall rules are:

1. Source TCP/UDP port
2. Destination TCP/UDP port

It is worth noting that it is impossible to allow and deny packets based on the process but instead on the port number. If you want the web browser to access the web, you must allow the respective ports, such as ports 80 and 443. This limitation differs from MS Windows' built-in firewall, which can restrict and allow traffic per application.

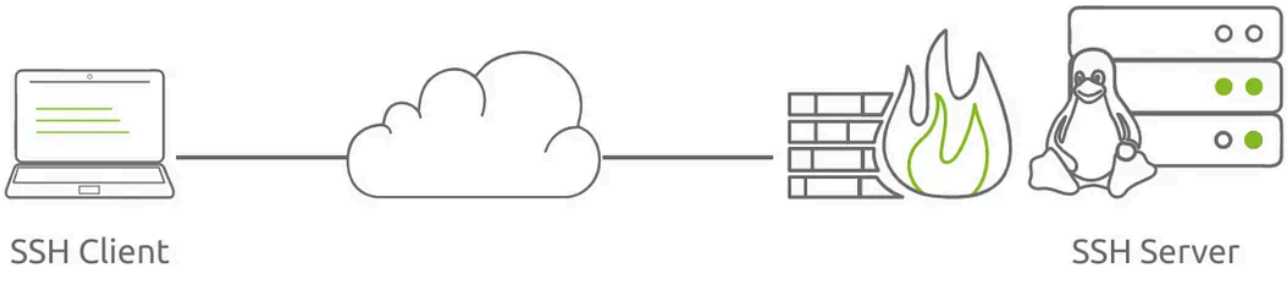
On a Linux system, a solution such as SELinux or AppArmor can be used for more granular control over processes and their network access. For example, we can allow only the `/usr/bin/apache2` binary to use ports 80 and 443 while preventing any other binary from doing so on the underlying system. Both tools enforce access control policies based on the specific process or binary, providing a more comprehensive way to secure a Linux system.

Let's look take a closer look at the different available Linux firewalls.

Netfilter

At the very core, we have netfilter. The netfilter project provides packet-filtering software for the Linux kernel 2.4.x and later versions. The netfilter hooks require a front-end such as `iptables` or `nftables` to manage.

In the following examples, we use different front-ends to netfilter in order to allow incoming SSH connections to the SSH server on our Linux system. As shown in the figure below, we want our SSH server to be accessible to anyone on the Internet with an SSH client.



iptables

As a front-end, iptables provides the user-space command line tools to configure the packet filtering rule set using the netfilter hooks. For filtering the traffic, iptables has the following default chains:

- **Input**: This chain applies to the packets incoming to the firewall.
- **Output**: This chain applies to the packets outgoing from the firewall.
- **Forward** This chain applies to the packets routed through the system.

Let's say that we want to be able to access the SSH server on our system remotely. For the SSH server to be able to communicate with the world, we need two things:

1. Accept incoming packets to TCP port 22.
2. Accept outgoing packets from TCP port 22.

Let's translate the above two requirements into `iptables` commands:

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

- `-A INPUT` appends to the INPUT chain, i.e., packets destined for the system.
- `-p tcp --dport 22` applies to TCP protocol with destination port 22.
- `-j ACCEPT` specifies (jump to) target rule ACCEPT.

```
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT
```

- `-A OUTPUT` append to the OUTPUT chain, i.e., packets leaving the system.
- `-p tcp --sport 22` applies to TCP protocol with source port 22.

Let's say you only want to allow traffic to the local SSH server and block everything else. In this case, you need to add two more rules to set the default behaviour of your firewall:

- `iptables -A INPUT -j DROP` to block all incoming traffic not allowed in previous rules.
- `iptables -A OUTPUT -j DROP` to block all outgoing traffic not allowed in previous rules.

In brief, the rules below need to be applied in the following order:

AttackBox Terminal

```
root@AttackBox# iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT
iptables -A INPUT -j DROP
iptables -A OUTPUT -j DROP
```

In practice, you should flush (delete) previous rules before applying new ones. This action can be achieved using `iptables -F`.

nftables

nftables is supported in Kernel 3.13 and later, adding various improvements over iptables, particularly in scalability and performance.

We will create a simple nftables configuration that allows traffic to our local SSH server.

Unlike iptables, nftables start with no tables or chains. We need to add the necessary tables and chains before adding rules. To begin, we will create a table, `fwfilter`.

```
nft add table fwfilter
```

- `add` is used to add a table. Other commands include `delete` to delete a table, `list` to list the chains and rules in a table, and `flush` to clear all chains and rules from a table.
- `table TABLE_NAME` is used to specify the name of the table we want to create or work on.

In our newly created table, `fwfilter`, we will add an *input* chain and an *output* chain for incoming and outgoing packets, respectively.

- `nft add chain fwfilter fwinput { type filter hook input priority 0 \; }`
- `nft add chain fwfilter fwoutput { type filter hook output priority 0 \; }`

The above two commands add two chains to the table `fwfilter`:

- `fwinput` is the input chain. It is of type `filter` and applies to the input hook.
- `fwoutput` is the output chain. It is of type `filter` and applies to the output hook.

With the two chains created within our table, we can add the necessary rule to allow SSH traffic. The following two rules are added to the table `fwfilter` to the chains `fwinput` and `fwoutput`, respectively:

- `nft add fwfilter fwinput tcp dport 22 accept` accepts TCP traffic to the local system's destination port 22.
- `nft add fwfilter fwoutput tcp sport 22 accept` accepts TCP traffic from the local system's source port 22.

We can check the shape of the `fwfilter` table using the command `nft list table fwfilter`:

AttackBox Terminal

```
root@AttackBox# sudo nft list table fwfilter
table ip fwfilter {
    chain fwinput {
        type filter hook input priority filter;
        tcp dport 22 accept
    }

    chain fwoutput {
        type filter hook output priority filter;
        tcp sport 22 accept
    }
}
```

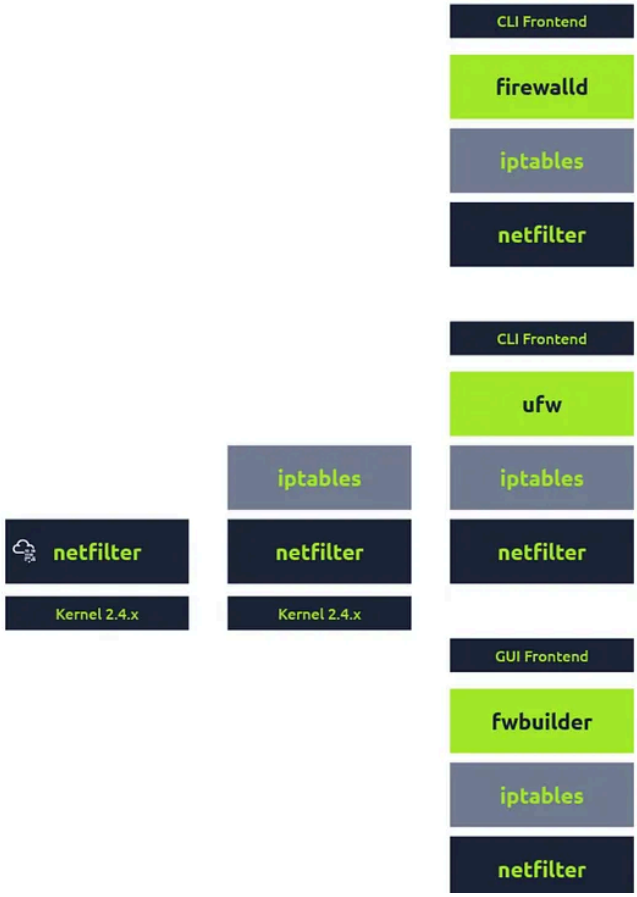
We hope the above example gave you a general overview of how nftables work.

UFW

After this overview of iptables and nftables, you might have started to develop the impression that configuring firewalls on Linux is a cumbersome, error-prone process. We already mentioned that iptables is like a front-end to netfilter; however, we can simplify things by providing a front-end to the front-end!

Example front-ends to iptables are shown in the figure below and can be divided into:

- Command-line Interface (CLI) front-ends, such as firewalld and ufw
- Graphical User Interface (GUI) front-ends, such as fwbuilder



UFW stands for uncomplicated firewall. Let's see how it stands for its promise of being *uncomplicated*. We will allow SSH traffic. This firewall rule can be achieved through one of the following commands:

```
ufw allow 22/tcp
```

It configures the firewall to `allow` traffic to TCP port 22. We can confirm our settings with the command `ufw status`.

AttackBox Terminal

```
root@AttackBox# sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
```

Firewall Policy

Before configuring a firewall, you need to decide upon the firewall policy. You might be the decision maker regarding the firewall policy or an enforcer of an existing security policy that covers firewall configuration. It all depends on the system you are protecting.

We will not go into security policies as this is outside the scope of this room. We will mention that the two main approaches are:

- Block everything and allow certain exceptions.
- Allow everything and block certain exceptions.

Each of the above two approaches has its advantages and disadvantages. Blocking everything with a limited set of exceptions would provide tighter and better security; however, it might cause inconvenience to the users depending on the situation.

Let's consider the following example. You are responsible for configuring the (host) firewall installed on the university computers. In this example, the academic institution has decided to block all outgoing and incoming traffic except for DNS, HTTP, and HTTPS traffic. In firewall terms, that's allowing UDP port 53 and TCP ports 80 and 443. This policy should allow browsing the Internet over HTTP and HTTPS; however, if one of the websites uses a non-standard HTTP or HTTPS port, it will be blocked. Dealing with these exceptions will create a challenge; keeping the firewall rules organised and properly documented is tricky as the number of exceptions grows over time.

There is a firewall running on the Linux VM. It is allowing port 22 TCP as we can ssh into the machine. It is allowing another TCP port; what is it?

```
tryhackme@ip-10-10-78-21:~/myvault$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
14298/udp                  ALLOW       Anywhere
12526/tcp                  ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
14298/udp (v6)             ALLOW       Anywhere (v6)
12526/tcp (v6)             ALLOW       Anywhere (v6)
```

Answer: **12526**

What is the allowed UDP port?

Answer: **14298**

Task 5: **Remote Access**

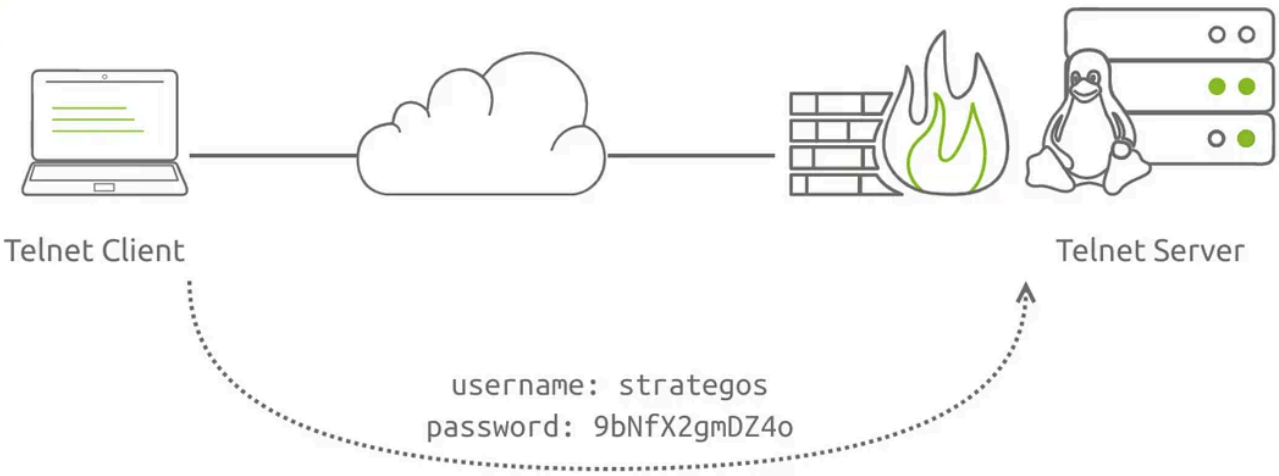
Providing remote access to a system is a very convenient way to access your system and files when you are not physically present at the target system's keyboard. However, this also means that you are voluntarily providing a service that attackers will target. Common attacks include:

1. Password sniffing
2. Password guessing and brute-forcing
3. Exploiting the listening service

Protecting Against Password Sniffing

Remote access can be achieved through many different protocols and services. Although all modern systems use encrypted protocols, such as the SSH protocol, for remote access, older systems might still use cleartext protocols, such as the Telnet protocol.

In the following figure, although the user has selected a strong password, it is being sent in cleartext, which is readable to anyone with a packet-capturing tool across the network path.

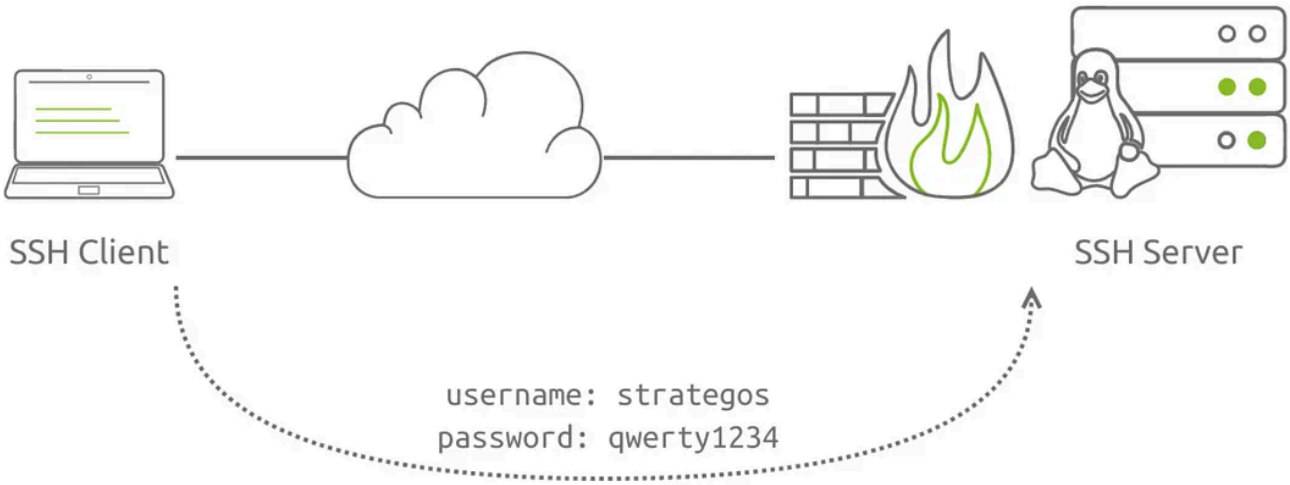


It is crucial to ensure that you select a protocol that encrypts traffic. The SSH protocol has been around for more than two decades. It has stood the test of time. It has many uses ranging from secure remote access to secure file transfers.

Protecting Against Password Guessing

When you set up your Linux system with SSH for remote administration, you also make your Linux box available for all interested parties. Many malicious hackers search the Internet for listening SSH servers and start to guess the login credentials; usually, they try `root` with the most common passwords.

The figure below shows that the system uses the SSH protocol to ensure encrypted communications; however, authentication relies on login credentials. Many users are tempted to use weak passwords or reuse the same password with other services. Although `qwerty1234` is not in an English dictionary, it is commonly found among the top 10 or 20 most common passwords, making it easy to guess.



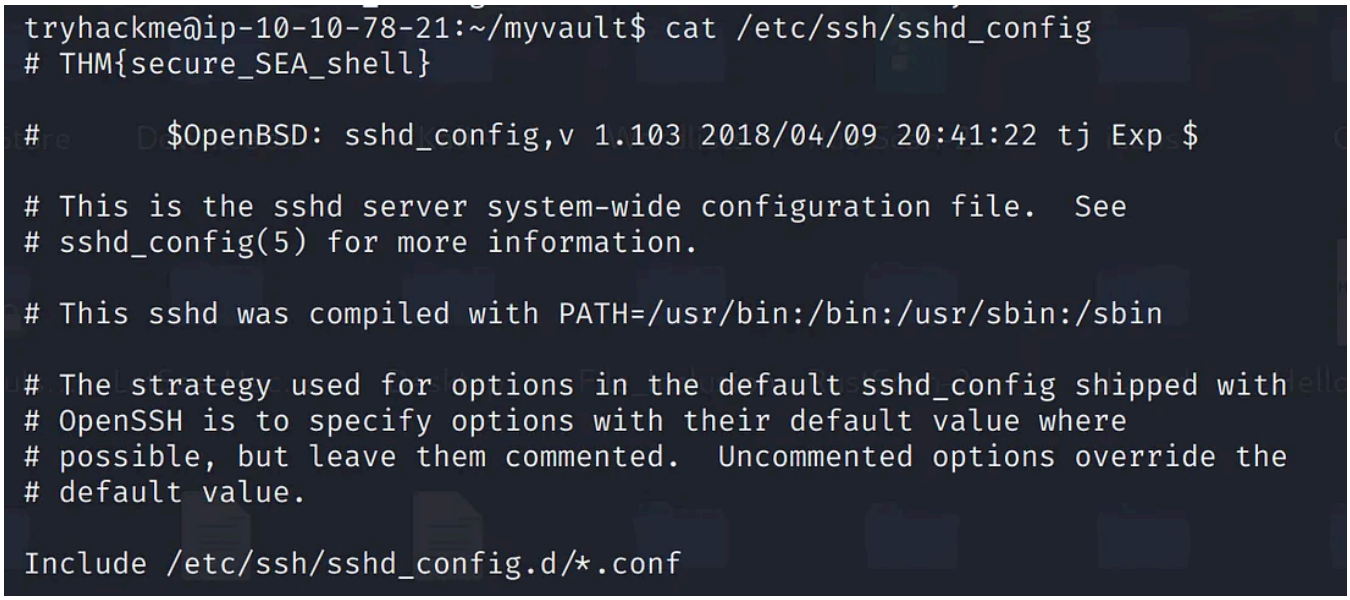
Because your SSH server will be configured to listen for incoming connections 24 hours a day, 365 days a year, evil users have all the time in the world to attempt one password after another. There are a few guidelines that you can use:

1. Disable remote login as `root`; force login as non-root users.
2. Disable password authentication; force public key authentication instead.

The reasoning behind the above guidelines is that you don't want the adversary to be able to attack the `root` account directly. Moreover, even if it is a non-root account, you don't want the attacker to gain access if there is a weakness in the password.

> The configuration of the OpenSSH server can be controlled via the `sshd_config` file, usually located at `/etc/ssh/sshd_config`. You can disable the root login by adding the following line:
>
> `PermitRootLogin no`
>
> Although a password such as `9bNfX2gmDZ4o` is difficult to guess, most users find memorising it inconvenient. Imagine if the account belongs to the *sudoers* (`sudo` group), and the user needs to type this password every time they need to issue a command with `sudo`. You may have to discipline to do that, but you cannot expect this to work for everyone.
>
> Many users are tempted to select a user-friendly password or share the same password across multiple accounts. Either approach would make the password easier for the attacker to guess.
>
> It would be best to rely on public key authentication with SSH to help improve the security of the remote login system and make it as fail-proof as possible.
>
> If you haven't created an SSH key pair, you must issue the command `ssh-keygen -t rsa`. It will generate a private key saved in `id_rsa` and a public key saved in `id_rsa.pub`.
>
> For the SSH server to authenticate you using your public key instead of your passwords, your public key needs to be copied to the target SSH server. An easy way to do it would be by issuing the command `ssh-copy-id username@server` where `username` is your username, and `server` is the hostname or IP address of the SSH server.
>
> It is best to ensure you have access to the physical terminal before you disable password authentication to avoid locking yourself out. You might need to ensure having the following two lines in your `sshd_config` file.
>
> - `PubkeyAuthentication yes` to enable public key authentication
> - `PasswordAuthentication no` to disable password authentication

What flag is hidden in the `sshd_config` file?



Answer: **THM{secure_SEA_shell}**

Task 6: **Securing User Accounts**

The `root` account carries with it tremendous power and hence risk. You are at risk of rendering your system unbootable with a simple mistake. Using a non-root account for everyday work is recommended to avoid sabotaging your system. However, `root` privileges are still needed for system maintenance, installing/removing software packages, and updating/configuring the system.

Use sudo

To avoid logging in as `root`, the better approach would be to have an account -created for administrative purposes- added to the *sudoers*, i.e. group who can use the `sudo` command. `sudo` stands for Super User Do and it should precede any command that requires `root` privileges.

Depending on the Linux distribution, we can add a user to the sudoers group in the following ways. Some distributions, such as Debian and Ubuntu, call the sudoers group `sudo`. In this case, you would need to issue the following command:

```
usermod -aG sudo username
```

- `usermod` modifies a user account.
- `-aG` appends to group.
- `sudo` is the name of the group of users who can use `sudo` on Debian-based distributions.
- `username` is the name of the user account you want to modify.

Other distributions, such as RedHat and Fedora, refer to the sudoers group as `wheel`. Consequently, you would need to issue the following command:

```
usermod -aG wheel username
```

The only difference is the name of the sudoers group.

Disable root

Once you have created an account for administrative purposes and added it to the `sudo` / `wheel` group, you might consider disabling the `root` account. A straightforward way is to modify the `/etc/passwd` and change the `root` shell to `/sbin/nologin`. In other words, edit `/etc/passwd` and change the line `root:x:0:0:root:/root:/bin/bash` to `root:x:0:0:root:/root:/sbin/nologin`.

Enforce a Strong Password Policy

The `libpwquality` library provides many options for password constraints. The configuration file can be found at:

- `/etc/security/pwquality.conf` on RedHat and Fedora
- `/etc/pam.d/common-password` on Debian and Ubuntu. You can install it using `apt-get install libpam-pwquality`

Here are a few example options:

- `difok` allows you to specify the number of characters in the new password that were not present in the old password.
- `minlen` sets the minimum allowed length for new passwords.
- `minclass` specifies the minimum number of required classes of characters; a class can be uppercase, lowercase, and digits, among others.
- `badwords` provides a space-separated list of words that must not be contained in the chosen password.
- `retry=N` prompts the user `N` times before returning an error.

Below is an example of `/etc/security/pwquality.conf`.

AttackBox Terminal

```
root@AttackBox# sudo cat /etc/security/pwquality.conf

difok=5
minlen=10
minclass=3
retry=2
```

You can check all the available options on the man page, `man pwquality.conf`.

Disable Unused Accounts

As part of system maintenance, it is vital to disable user accounts that no longer need access to the system in question. For instance, these users might have moved to another department or quit the company.

You can disable a user account in the same way we would disable the root account. An easy way would be to edit the `/etc/passwd` file and set the shell of the user account we want to disable to `/sbin/nologin`.

Let's say that we want to disable the account of the user Michael with username `michael`.

- Enabled account: `michael:x:1000:1000:Michael:/home/michael:/usr/bin/fish`
- Disabled account: `michael:x:1000:1000:Michael:/home/michael:/sbin/nologin`

We should do the same for local services. In other words, we should set the shell to `sbin/nologin` for all the local service accounts such as `www-data`, `mongo`, and `nginx`, to name a few. The reason is that these services need accounts to run on the system but would never need to log in and access a shell. Any of these services could perhaps have an RCE (Remote Code Execution) vulnerability, and by setting the shell to `nologin`, we can at least prevent interactive logins for the account of the affected service.

One way to disable an account is to edit the `passwd` file and change the account's shell. What is the suggested value to use for the shell?

Answer: **/sbin/nologin**

What is the name of the RedHat and Fedora systems sudoers group?

Answer: **wheel**

What is the name of the sudoers group on Debian and Ubuntu systems?

Answer: **sudo**

Other than `tryhackme` and `ubuntu`, what is the username that belongs to the sudoers group?

```
tryhackme@ip-10-10-78-21:~/myvault$ cat /etc/group | grep sudo
sudo:x:27:ubuntu,tryhackme,blacksmith
tryhackme@ip-10-10-78-21:~/myvault$
```

Answer: **blacksmith**

Task 7: **Software and Services**

Every piece of software you install on your system also increases the number of potential vulnerabilities. In other words, installing additional software packages and new services increases the vulnerabilities an attacker can exploit to gain access to your system and, eventually, to other systems on your network. You can follow some guidelines to help you reduce the attack surface.

Disable Unnecessary Services

One of the easiest ways to improve your security posture is by removing or disabling unneeded services and packages. In simple terms, we need to minimise the number of installed system packages as every package carries some risk, and we cannot know when a related vulnerability will be discovered. The best policy is to avoid installing unneeded packages.

For example, if you don't need a web server, you should ensure you don't install one. If you needed to run a web server at one point but no longer need it now, you should remove it or at least disable it. Otherwise, you will be exposing yourself to unnecessary risk.

Block Unneeded Network Ports

After you remove any packages that are not required and disable preinstalled services that might not be removed, it is critical to set your firewall rules accordingly. If you don't have a web server, there is no reason to allow packets to TCP ports 80 and 443. The reasoning behind this is that if the attacker manages to start a disabled service, the firewall will block its traffic, and the attacker won't be able to access its TCP port(s).

Avoid Legacy Protocols

At one point in the past, Telnet was the primary protocol to remote access a system; the TFTP protocol was commonly used to transfer files. Such protocols should no longer be allowed as secure alternatives have been released.

Instead of Telnet, the SSH protocol is now widely available. For example, the Secure File Transfer Protocol (SFTP) protocol provides a great alternative to the TFTP protocol. The critical point is that a secure alternative is selected and used.

Remove Identification Strings

Whenever you connect to a remote server, it usually replies with its version number. This information would reveal various information to the attacker, such as the name of the server/program, the version number, and the host operating system.

Besides FTPS, what is another secure replacement for TFTP and FTP?

Answer: **SFTP**

Task 8: **Update and Upgrade Policies**

It is vital that you keep your system updated with the latest security patches and bug fixes.

You can update a Debian-based distribution, such as Ubuntu, with the following two commands:

1. `apt update` to download package information from the configured sources
2. `apt upgrade` to install available upgrades for all packages from the configured sources

You can update a RedHat or Fedora system using the following:

- `dnf update` on newer releases (Red Hat Enterprise Linux 8 and later)
- `yum update` on older releases (Red Hat Enterprise Linux 7 and earlier)

Since we are talking about production systems, we need to select distributions that will continue to receive updates smoothly for many years.

Ubuntu LTS Releases

For example, Ubuntu releases a Long Term Support (LTS) version every two years. LTS releases have an even version number (indicating the year) with 04 (for April), such as 18.04, 20.04, and 22.04. An LTS version will grant you five years of security updates for the base OS without a subscription and another five years if you pay for Extended Security Maintenance (ESM). For more information, you might want to check The Ubuntu lifecycle and release cadence.

Using an Ubuntu LTS release, you can download updates without a subscription for the first five years. Consider the following Linux system with Ubuntu 14.04. It was released in 2014, and its free support ended in April 2019. In other words, free update service was available till 2019. In the terminal below, we see that there are 133 additional updates that we can download and install if we have an Ubuntu Pro (Infra-Only) subscription (previously known as Ubuntu Advantage (UA) for Infrastructure).

```
root@AttackBox# user@tryhackme$ uname -a
Linux ubuntu-14-vm 4.4.0-148-generic #174~14.04.1-Ubuntu SMP Thu May 9 08:18:11 UTC 2019 i686 athlon i686 GNU/Linux

user@tryhackme$ sudo apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

133 additional updates are available with UA Infrastructure ESM.
To see these additional updates run: apt list --upgradable
See https://ubuntu.com/advantage or run: sudo ua status
```

Ubuntu 14.04 was released more than five years ago; hence, free updates are no longer available. From the security point of view, the user should either get an Ubuntu Pro subscription or switch to a new LTS release, such as Ubuntu 22.04. Continuing to run a system that does not receive security updates exposes the system and the whole network to additional risk.

RedHat Releases

RedHat Enterprise Linux 8 and 9 offer 12 years of support in three phases:

1. Full Support for five years
2. Maintenance Support for five years
3. Extended Life Phase for two years

For example, Red Hat Enterprise Linux 9 was released on May 18, 2022. It can benefit from updates as follows:

1. Full support till May 31, 2027
2. Maintenance support till May 31, 2032

For more details, check Red Hat Enterprise Linux Life Cycle.

Kernel Updates

Updating the system should not be limited to the installed software; it should consider updating the kernel. For instance, in 2016, a security vulnerability that affects the Linux kernel was discovered. It allows an attacker to gain root access to a system by exploiting a race condition in the copy-on-write (COW) mechanism, giving it the name “Dirty COW.” The vulnerability is present in all Linux kernel versions from 2.6.22 onwards. It has been patched in most major Linux distributions, but it is still a threat to systems that have not been updated.

Because Dirty COW is a severe vulnerability that can allow attackers to gain root access to Linux systems, it is crucial to keep your system, including its kernel, up-to-date with the latest security patches to reduce the risk of exploitation.

Automatic Updates

Now that you have a Linux system with security updates throughout its life, we must ensure that the updates are properly installed, and security fixes are applied.

- Stay updated with security news in case of a vulnerability that affects your systems is disclosed.
- Depending on the Linux distribution that you are using, consider configuring automatic updates. Automating updates on Linux distributions that prioritize stability over cutting-edge technology would be safe.

What command would you use to update an older Red Hat system?

Answer: **yum update**

What command would you use to update a modern Fedora system?

Answer: **dnf update**

What two commands are required to update a Debian system? (Connect the two commands with `&&`.)

Answer: **apt update && apt upgrade**

What does `yum` stand for?

Answer: **Yellowdog Updater, Modified**

What does `dnf` stand for?

Answer: **Dandified Yum**

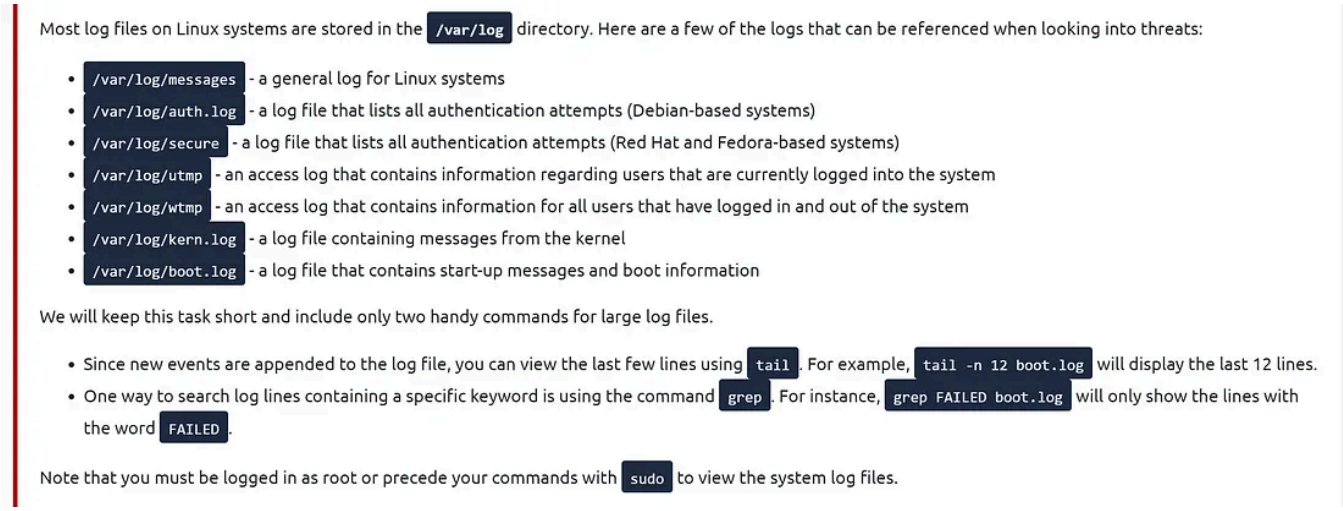
What flag is hidden in the `sources.list` file?

```
tryhackme@ip-10-10-78-21:~/myvault$ find / -type f -name sources.list 2>/dev/null
/snap/core/14946/etc/apt/sources.list
/usr/share/doc/apt/examples/sources.list
/etc/apt/sources.list
/var/lib/ubuntu-advantage/apt-esm/etc/apt/sources.list
tryhackme@ip-10-10-78-21:~/myvault$ cat /etc/apt/sources.list
# THM{not_Advanced_Persistent_Threat}

## Note, this file is written by cloud-init on first boot of an instance
## modifications made here will not survive a re-bundle.
## if you wish to make changes you can:
## a.) add 'apt_preserve_sources_list: true' to /etc/cloud/cloud.cfg
##     or do the same in user-data
## b.) add sources in /etc/apt/sources.list.d
## c.) make changes to template file /etc/cloud/templates/sources.list.tmpl
```

Answer: **THM{not_Advanced_Persistent_Threat}**

Task 9: **Audit and Log Configuration**



What command can you use to display the last 15 lines of `kern.log`?

Answer: **tail 15 kern.log**

What command can you use to display the lines containing the word `denied` in the file `secure`?

Answer: **grep denied secure**

Task 10: **Conclusion and Final Notes**

> This room has covered various topics related to hardening your Linux system. It is essential to follow some general guidelines. Most of them are common sense that a system administrator will develop after some experience. A minimal set of guidelines to adopt includes the following:
>
> - Document host information.
> - Apply and test the changes on a test system. Test before you make changes to production environments.
> - Document all changes carried out.
>
> The above guidelines can quickly extend over a few pages depending on the company size.

Tryhackme Walkthrough Tryhackme Writeup

Follow

Written by Daniel Schwarzentraub

116 Followers · 4 Following

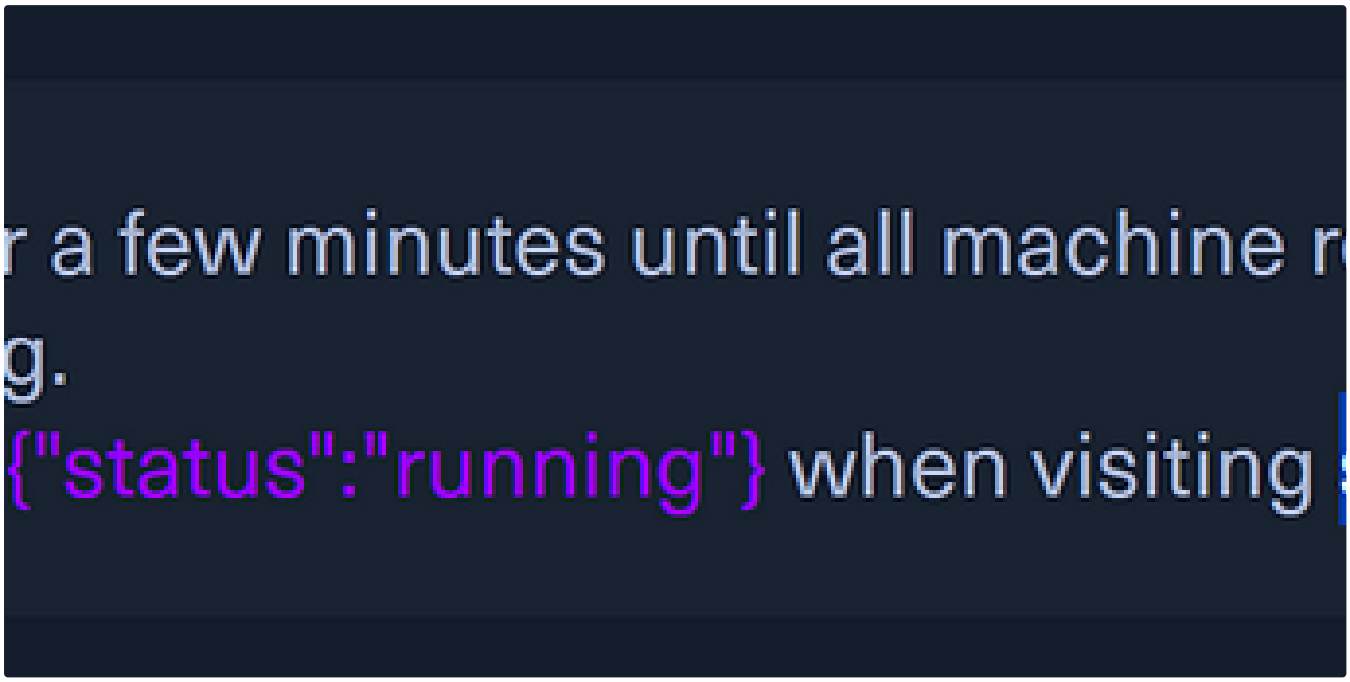
PNW_Hacker

No responses yet

What are your thoughts?

Respond

More from Daniel Schwarzentraub

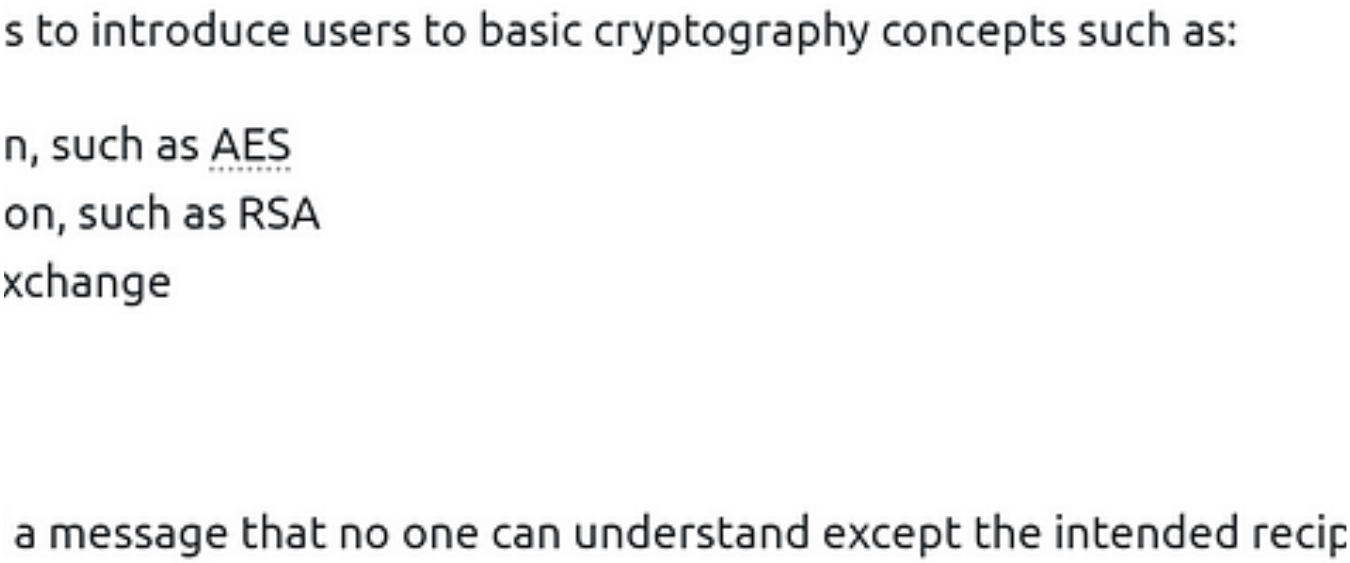


Daniel Schwarzentraub

HTB—Tier 1 Starting Point: Three

HTB—Tier 1 Starting Point: Three

Jul 20, 2023 4 2

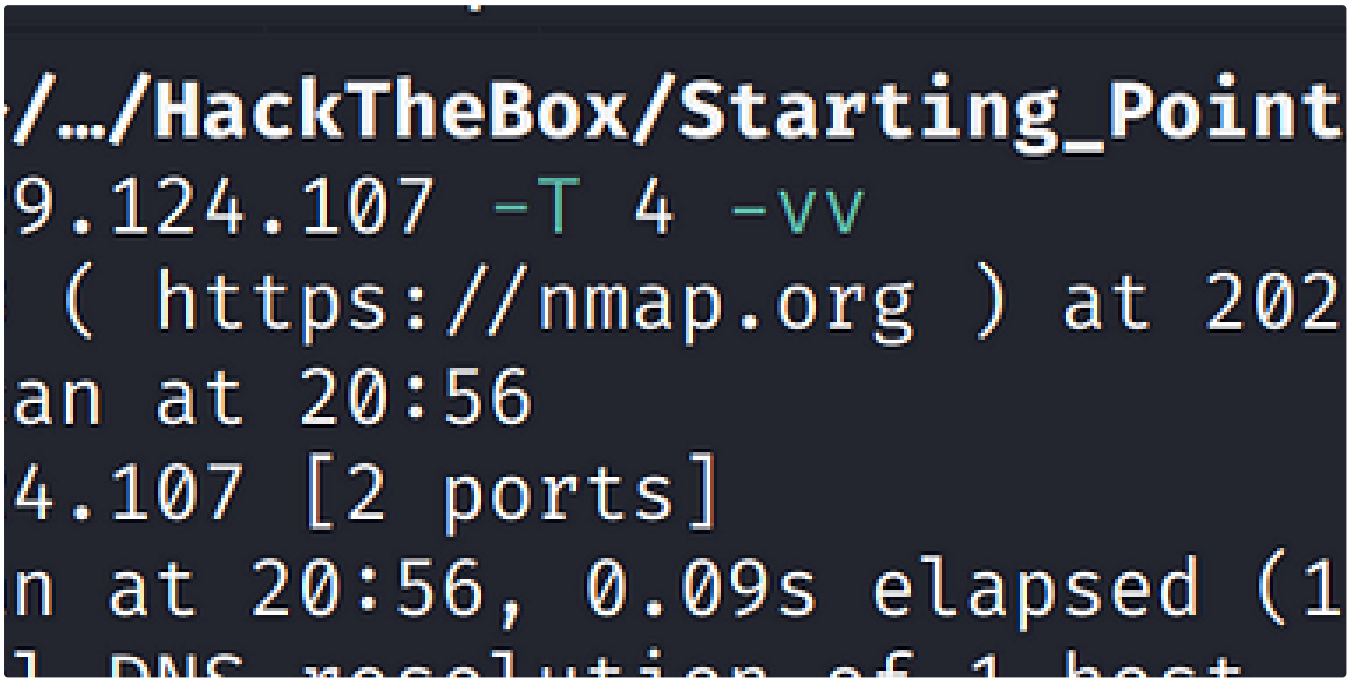


Daniel Schwarzentraub

Tryhackme: Introduction to Cryptography

Tryhackme: Introduction to Cryptography

Sep 26, 2023 2



Daniel Schwarzentraub

HTB—Tier 2 Starting Point: Archetype

HTB—Tier 2 Starting Point: Archetype

Jul 21, 2023

erative that we understand and can protect against common attacks.

mon techniques used by attackers to target people online. It will also teach some of the best wa

Daniel Schwarzentraub

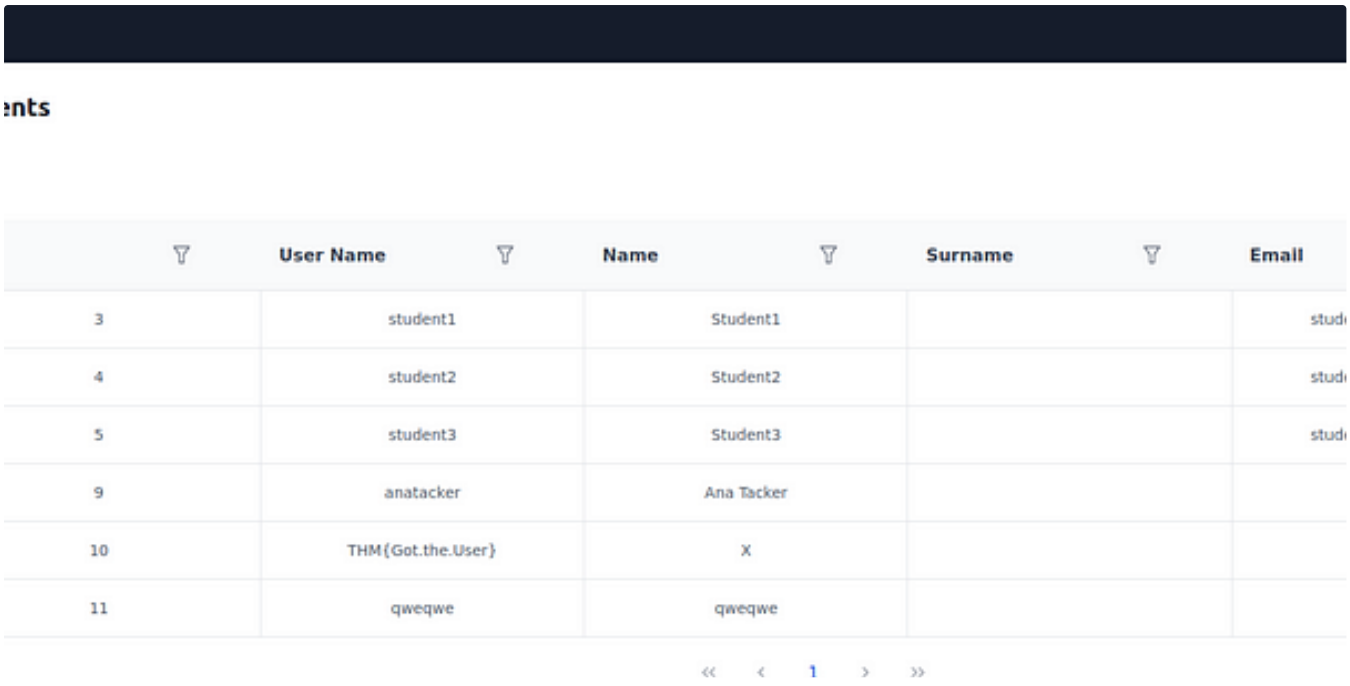
Tryhackme Free Walk-through Room: Common Attacks

Tryhackme Free Walk-through Room: Common Attacks

Sep 13, 2024

See all from Daniel Schwarzentraub

Recommended from Medium



ents

	User Name	Name	Surname	Email
3	student1	Student1		stud
4	student2	Student2		stud
5	student3	Student3		stud
9	anatacker	Ana Tacker		
10	THM{Got.the.User}	X		
11	qweqwe	qweqwe		

« ‹ 1 › »

embossdotar

TryHackMe—Session Management—Writeup

Key points: Session Management | Authentication | Authorisation | Session Management Lifecycle | Exploit of vulnerable session management...

Aug 7, 2024 27

beyza

Tryhackme: Crocc Crew Write Up

CTF Write-Up: Crocc Crew Port Scan Results:

Aug 27, 2024

Lists

Staff picks

796 stories · 1560 saves

Stories to Help You Level-Up at Work

19 stories · 912 saves

Self-Improvement 101

20 stories · 3196 saves

Productivity 101

20 stories · 2707 saves

In T3CH by Axoloth

TryHackMe | Deja Vu | WriteUp

Exploit a recent code injection vulnerability to take over a website full of cute dog pictures!

Oct 13, 2024 50

In T3CH by Axoloth

TryHackMe | Training Impact on Teams | WriteUp

Discover the impact of training on teams and organisations

Nov 5, 2024 60

In T3CH by Axoloth

TryHackMe | K8s Runtime Security | WriteUp

Secure a Kubernetes environment using in-house offerings and runtime security tools like Falco.

Sep 15, 2024 50

Mohamed Ali

TryHackMe—Cluster Hardening—Writeup

Learn initial security considerations when creating a Kubernetes cluster.

Jul 25, 2024