System Hardening Protocol

Made by: Tedi Dobi

The objective of this project is to implement and document the process of hardening an Ubuntu-based system running within a VMware virtual environment. System hardening is a critical aspect of cybersecurity, aimed at minimizing potential attack surfaces by securing system configurations, disabling unnecessary services, enforcing access controls, and applying recommended security policies.

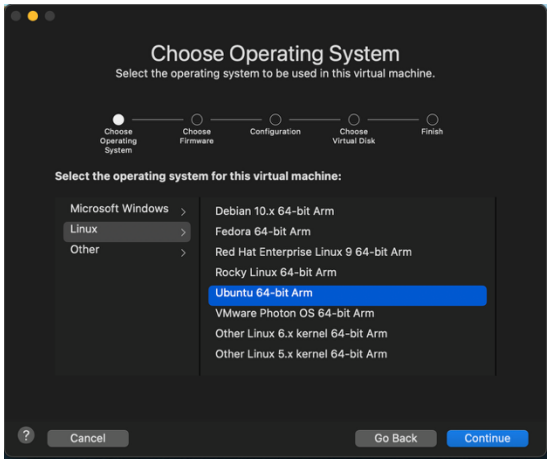
In this project, Ubuntu serves as the target operating system due to its wide usage in both server and desktop environments, while VMware provides a flexible and isolated platform for testing security measures without affecting production systems. The hardening process follows established best practices and guidelines, including user privilege management, firewall setup, AppArmor, system logging and system update policies.

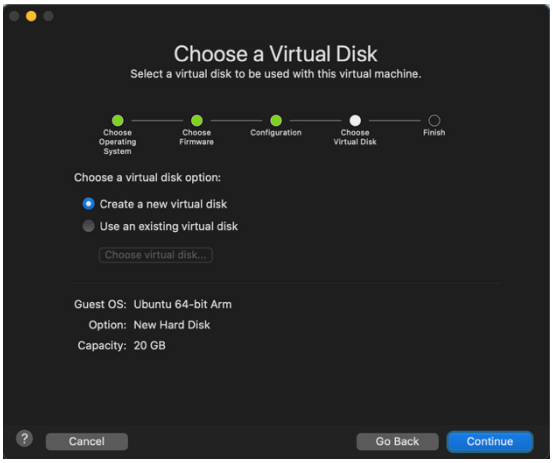
By the end of the project, the system will meet a higher standard of security and resilience, better prepared to withstand common threats. The accompanying protocol outlines each step taken during the hardening process, providing both technical details and the rationale behind each decision.

1.1 Software Selection

For this project, I chose Ubuntu 24.04.2 LTS because it offers stability, consistent security updates and wide compatibility. The set up was done on VMware. The installation process included the following steps:

a. Downloaded the Ubuntu ISO image from the official website.
b. Initially selected Ubuntu 64-bit (ARM) as the guest operating system.
c. Mounted the ISO file as a virtual CD drive in the VM settings.
d. Upon booting, only the terminal interface was available.
e. Set the details: name, password, server name
f. Installed GUI packages (e.g., GNOME or another desktop environment) to enable a graphical interface for easier use.
g. Create a Snapshot





h. Reduced Packages:
- Unused desktop application: **sudo apt purge libreoffice* thunderbird transmission-gtk cheese rhythmbox gnome-mahjongg gnome-sudoku aisleriot**
- Printing Services**: sudo apt purge cups cups-browsed printer-driver-***
- Network Services Not Needed (disable discovery, remote desktop): **sudo apt purge avahi-daemon avahi-autoipd telnet xrdp**
- Development tools: **sudo apt purge gcc g++ make manpages-dev**

1.2 Review accounts, groups and privillegess

The first user, who was created during the setup of the VM, is *tedi1*. Firstly, I started by configuring the password criterias using **pwquality**, enabling the lockout feature using **faillock** and setting up 2FA time-based code using Google Authenticator. When this setup was done, I created the new user accounts, reviewed their privilleges and set the resources quota for each of them.

a. Set the minimum password length of 12 characters, at least 3 character types and check the password for words from the cracklib dictionary.
Commands:
   - Install "pwquality" module: **sudo apt install pwquality**
   - Open the configuration file: **sudo nano /etc/security/pwquality.conf**
   - Uncomment the line **"#minlen = 8**" and replace 8 with 12.
   - Uncomment the line **"#minclass = 3"**: there should be at least 3 character types.
   - Uncomment the line **"#minclass = 3"**: there should be at least 3 character types.
   - To test if the changes have been applied, you can try to change the current user's password. The password can be changed using **"sudo passwd *username*"**

```
# Minimum acceptable size for the new password (plus one if
# credits are not disabled which is the default). (See pam_cracklib manual.)
# Cannot be set to lower value than 6.
minlen = 12
```

```
# The minimum number of required classes of characters for the new
# password (digits, uppercase, lowercase, others).
minclass = 3
```

```
# Whether to check for the words from the cracklib dictionary.
# The check is enabled if the value is not 0.
dictcheck = 1
```

b. Change default permission settings for home directories, so that they get locked down on creation and there is no need to use **"sudo chmod 700 *"** for each user's home directory.
Command:
   - Go to **/etc/login.defs**
   - Look for the line **UMASK 022**
   - Change it to **UMASK 077**
   - This change applies only to new accounts, not to the already created, so use **"sudo chmod 700 *"** to also change the permissions for the rest of the accounts

```
#
ERASECHAR       0177
KILLCHAR        025
UMASK           077
```

c. In order to provide good security and prevent brute force attacks, I enabled the lockout property after 3 failed login attempts. This is done by configuring *faillock* according to the CIS Benchmark for Ubuntu LTS. Commands:
   - First go to /etc/security/faillock.conf and uncomment the following lines:
     deny = 3
     fail_interval = 900

   unlock_time = 600

```
# Deny access if the number of consecutive authentication failures
# for this user during the recent interval exceeds n tries.
# The default is 3.
deny = 3
```

```
# The length of the interval during which the consecutive
# authentication failures must happen for the user account
# lock out is <replaceable>n</replaceable> seconds.
# The default is 900 (15 minutes).
fail_interval = 900
```

```
# The access will be re-enabled after n seconds after the lock out.
# The value 0 has the same meaning as value `never` - the access
# will not be re-enabled without resetting the faillock
# entries by the `faillock` command.
# The default is 600 (10 minutes).
unlock_time = 600
```

- Then go to /etc/pam.d/common-auth
- Add the following lines:

auth required pam_faillock.so preauth
auth [default=die] pam_faillock.so authfail
auth sufficient pam_faillock.so authsucc

- Make sure the following lines are in this exact order, surrounding the pam_unix.so line.

```
 GNU nano 7.2                    /etc/pam.d/common-auth
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.

#Added to enable faillock
auth    required                        pam_faillock.so preauth

# here are the per-package modules (the "Primary" block)
auth    [success=2 default=ignore]      pam_unix.so nullok
auth    [success=1 default=ignore]      pam_sss.so use_first_pass

# Added to enable faillock
auth    [default=die]                   pam_faillock.so authfail
auth    sufficient                      pam_faillock.so authsucc

# here's the fallback if no module succeeds
auth    requisite                       pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
```

- Then go to /etc/pam.d/common-account and add the following line at the end of the file:
  account required pam_faillock.so

```
  GNU nano 7.2                  /etc/pam.d/common-account
# pam-auth-update(8) for details.
#

# here are the per-package modules (the "Primary" block)
account [success=1 new_authtok_reqd=done default=ignore]        pam_unix.so
# here's the fallback if no module succeeds
account requisite                       pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
account required                        pam_permit.so
# and here are more per-package modules (the "Additional" block)
account sufficient                      pam_localuser.so
account [default=bad success=ok user_unknown=ignore]    pam_sss.so

# Added to enable faillock
account required pam_faillock.so

# end of pam-auth-update config
```

To test what I have done until now, I created a new user “testuser”:

- **sudo useradd -m -d /home/testuser -s /bin/bash testuser**
- Run the following command to see the permissions: “ **ls -ld /home/testuser**” . It returns drwx ----- 2

```
tedi1@tedi1server:/$ ls -ld /home/testuser
drwx------ 2 testuser testuser 4096 May 30 21:23 /home/testuser
```

- Then I tested the implemented criterias for setting a password for the testuser.

```
tedi1@tedi1server:/$ sudo passwd testuser
New password:
BAD PASSWORD: The password is shorter than 12 characters
Retype new password:
Sorry, passwords do not match.
New password:
BAD PASSWORD: The password contains less than 3 character classes
Retype new password:
Sorry, passwords do not match.
New password:
BAD PASSWORD: The password is shorter than 12 characters
Retype new password:
Sorry, passwords do not match.
passwd: Have exhausted maximum number of retries for service
passwd: password unchanged
tedi1@tedi1server:/$
```

```
tedi1@tedi1server:/$ sudo passwd testuser
New password:
Retype new password:
passwd: password updated successfully
tedi1@tedi1server:/$
```

- Then I tested the faillock feature with deny set to 3. This test was done by trying to ssh into “testuser”. First, I downloaded the sshd package, then tried “**ssh testuser@192.168.8.248**” and entered the wrong password three times. When checking the failed password attempts, all of the previous 3 attemps were counted. When trying immediately once again to ssh into testuser and entering the correct password, permission was still denied due to faillock, which proved that faillock is functional.

```
tedi1@tedi1server:~$ ssh testuser@192.168.8.248
testuser@192.168.8.248's password:
Permission denied, please try again.
testuser@192.168.8.248's password:
Permission denied, please try again.
testuser@192.168.8.248's password:
testuser@192.168.8.248: Permission denied (publickey,password).
tedi1@tedi1server:~$ sudo faillock --user testuser
[sudo] password for tedi1:
testuser:
When                Type  Source                                           Valid
2025-05-31 14:18:22 RHOST 192.168.8.248                                       V
2025-05-31 14:18:26 RHOST 192.168.8.248                                       V
2025-05-31 14:18:33 RHOST 192.168.8.248                                       V
```

d. After the above configurations, I decided to also add 2FA as a security measure, by implementing a time-based code from Google Authenticator. “sudo” is not included in the setup, in case something goes wrong.

- Create copies of common-auth, common-account and faillock.conf, just in case something goes wrong.
  sudo cp /etc/pam.d/common-auth /etc/pam.d/common-auth.bak
  sudo cp /etc/pam.d/common-account /etc/pam.d/common-account.bak
  sudo cp /etc/security/faillock.conf /etc/security/faillock.conf.bak

- Install the Google Authenticator module: **sudo apt install libpam-google-authenticator**
- Go to /etc/pam.d/common-auth and add the following on the very top:
  auth required pam_google_authenticator.so nullok

```
# Added to enable Google Authenticator
auth    required                        pam_google_authenticator.so nullok

# Added to enable faillock
auth    required                        pam_faillock.so preauth

# here are the per-package modules (the "Primary" block)
auth    [success=2 default=ignore]      pam_unix.so nullok
auth    [success=1 default=ignore]      pam_sss.so use_first_pass

# Added to enable faillock
auth    [default=die]                   pam_faillock.so authfail
auth    sufficient                      pam_faillock.so authsucc
```

- Now switch user and go to the “testuser”
- Once you have switched to testuser, enter the following:
  google-authenticator
  Do you want your authentication tokes to be time-based ? (y / n) -> Enter y

- Copy the generated code, paste it into the Google Authenticator app and enter the new generated code.
- The above steps should be repeated for all users in order to set up time-based one time code from Google Authenticator. The screenshots below show the setup of Google Authenticator.

```
tedi1@tedi1server:~$ sudo nano /etc/pam.d/common-auth
tedi1@tedi1server:~$ su - testuser
Password:
testuser@tedi1server:~$ google-authenticator

Do you want authentication tokens to be time-based (y/n) y
Warning: pasting the following URL into your browser exposes the OTP secret to Google:
```

```
Your new secret key is:
Enter code from app (-1 to skip): 417752
Code confirmed
Your emergency scratch codes are:




Do you want me to update your "/home/testuser/.google_authenticator" file? (y/n) y

Do you want to disallow multiple uses of the same authentication
token? This restricts you to one login about every 30s, but it increases
your chances to notice or even prevent man-in-the-middle attacks (y/n)
```

```
Do you want me to update your "/home/testuser/.google_authenticator" file? (y/n) y

Do you want to disallow multiple uses of the same authentication
token? This restricts you to one login about every 30s, but it increases
your chances to notice or even prevent man-in-the-middle attacks (y/n) y

By default, a new token is generated every 30 seconds by the mobile app.
In order to compensate for possible time-skew between the client and the server,
we allow an extra token before and after the current time. This allows for a
time skew of up to 30 seconds between authentication server and client. If you
experience problems with poor time synchronization, you can increase the window
from its default size of 3 permitted codes (one previous code, the current
code, the next code) to 17 permitted codes (the 8 previous codes, the current
code, and the 8 next codes). This will permit for a time skew of up to 4 minutes
between client and server.
Do you want to do so? (y/n) n

If the computer that you are logging into isn't hardened against brute-force
login attempts, you can enable rate-limiting for the authentication module.
By default, this limits attackers to no more than 3 login attempts every 30s.
Do you want to enable rate-limiting? (y/n) y
testuser@tedi1server:~$
```

- Now, when trying to log in to “testuser”, we are required both the verification code from Google Authenticator and the password.

```
tedi1@tedi1server:~$ su - testuser
Verification code:
Password:
testuser@tedi1server:~$
```

e. Now that the account security setup is done, it is time to create the user accounts and set their privilleges. For this project, I have created the following accounts, set the password according to the above criteria and enabled Google-Authenticator for all of them:

- **admin:** Main administartor, full controll and fallback access, has sudo privileges
- **maint:** Maintenance user with limited sudo, used for maintenance
- **appuser:** Used to run the web server
- **auditor:** Can view the system logs
- **backup:** Has read-only access for backup tasks
- **limited:** Regular unprivilleged user

```
tedi1@tedi1server:/$ sudo useradd -m -d /home/admin -s /bin/bash admin
tedi1@tedi1server:/$ sudo passwd admin
New password:
Retype new password:
passwd: password updated successfully
tedi1@tedi1server:/$ sudo usermod -aG sudo admin
```

```
tedi1@tedi1server:/$ su - admin
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

admin@tedi1server:~$ google-authenticator

Do you want authentication tokens to be time-based (y/n) y
Warning: pasting the following URL into your browser exposes the OTP secret to G
oogle:
```

- Following this, I added the users to the correct groups, edited the “visudo” file and set the privileges so that each user can perform his task and nothing more.

  admin: admin, adm, sudo
  maint: maint, sudo
  auditor: auditor, adm
  limited: limited

  In sudoers (sudo visudo)
  maint ALL=(ALL) NOPASSWD: /bin/systemctl restart apache2, /bin/systemctl status apache2
  appuser ALL=(ALL) !ALL

```
# Restrict maint only to web server and logs only
maint   ALL=(ALL) NOPASSWD: /bin/systemctl restart apache2, /bin/systemctl status apache2

# Deny sudo for appuser (Application Service User)
appuser ALL=(ALL) !ALL
```

f. Next is the configuration of the maximum number of opened files, memory usage limit and disk space quotas per user.

Commands for maximum number of open files and memory usage limits:

- Go to limits.conf: sudo nano /etc/security/limits.conf
- Make the following address space configurations for testuser, maint and admin
- The go to /etc/pam.d/limits.so and add the following line at the end of the file: **session required pam_limits.so**. This line will enforce the limits set in the limits.conf file.

```
# Limits for maint
maint           soft    nofile          1024
maint           hard    nofile          2048
maint           soft    as              524288
maint           hard    as              1048576

# Limits for admin
admin           soft    nofile          2048
admin           hard    nofile          4096
admin           soft    as              1048576
admin           hard    as              2097152
```

```
# Limits for auditor
auditor         soft    nofile          256
auditor         hard    nofile          512
auditor         soft    as              131072
auditor         hard    as              262144

# Limits for backup
limited         soft    nofile          256
limited         hard    nofile          512
limited         soft    as              131072
limited         hard    as              262144
```

g. Next is the firewall setup. For my project I used the **uncomplicated firewall (ufw)**, which is already installed in the Ubuntu systems.

- By default the firewall is disabled, so I started by enabling it: **sudo ufw enable**
- For my project, I did not require any open ports apart from port 80, 443 and 514, which is used later for logging, therefore I did not add any othe rules.

```
tedi1@tedi1server:/$ sudo ufw allow 80
Rules updated
Rules updated (v6)
tedi1@tedi1server:/$ sudo ufw allow 443
Rules updated
Rules updated (v6)
tedi1@tedi1server:/$
```

1.3 Security Upgrades

This part states how the automatic security upgrades are implemented in Ubuntu LTS 22.04. Additionally, this section includes the configuration of a container update procedure, a prototype just to show understanding.

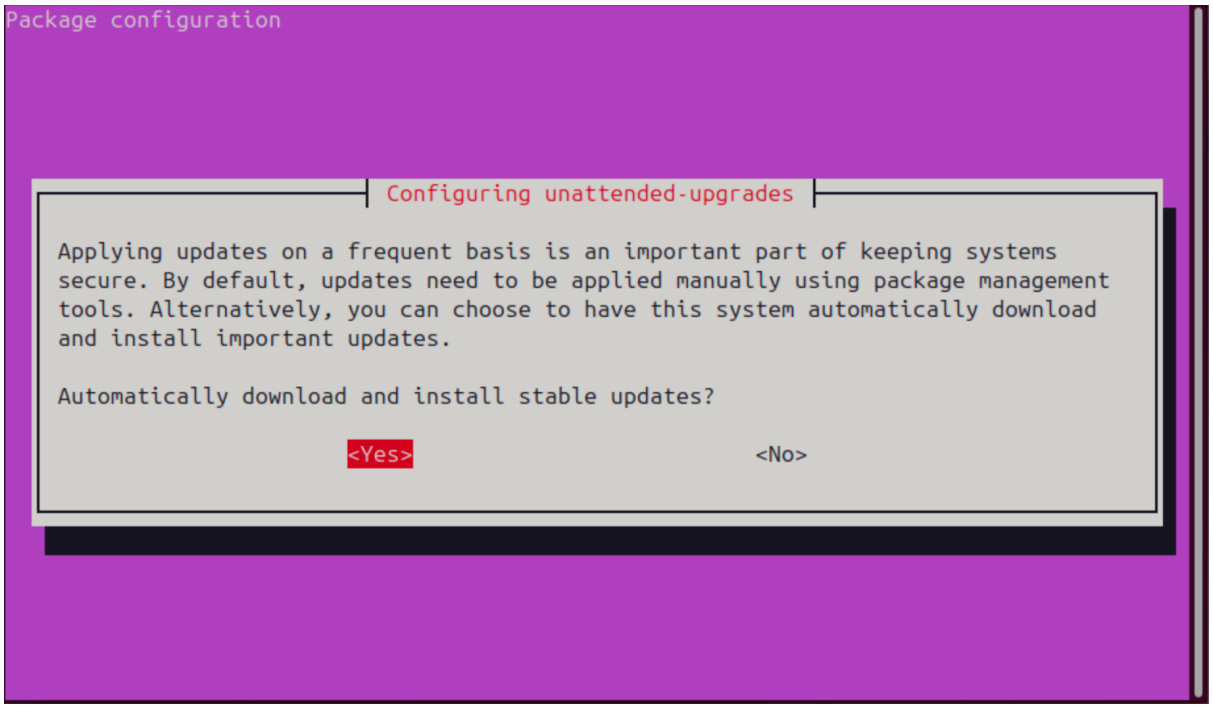
Automatic security upgrades
I configured unattended-upgrades on Ubuntu LTS to automatically install security updates daily. The periodic configuration ensures package lists are updated and cleaned, and critical security patches are applied without user intervention.

Prototype: Container Update Procedure
Although containers are not part of the primary system configuration, I created a prototype using Docker and Watchtower. A test container (nginx) was deployed and monitored by Watchtower, which was configured to check for updates every 60 seconds. A new image version was pulled manually to simulate an update, and Watchtower successfully restarted the container with the latest image. This prototype demonstrates how container-based systems can be kept updated automatically.

a. Automatic Security Upgrades
Commands:

- Download the following packages: **sudo apt install unattended-upgrades apt-listchanges**
- Then run the following command: **sudo dpkg-reconfigure --priority=low unattended-upgrades.** This will open a configuration prompt asking "Do you want to automatically download and install stable updates?", to which you should answer "Yes".

```
tedi1@tedi1server:/$ sudo apt install unattended-upgrades apt-listchanges
[sudo] password for tedi1:
Reading package lists... Done
```



- Then you should check to see if this has been applied properly. Sometimes, the user might have to update the file by himself. Go to **/etc/apt/apt.conf.d/10periodic**. In my case, I set the update to daily (1) and the clean up every week (7).

```
tedi1@tedi1server:/$ sudo cat /etc/apt/apt.conf.d/10periodic
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Download-Upgradeable-Packages "1";
APT::Periodic::AutocleanInterval "7";
APT::Periodic::Unattended-Upgrade "1";
```

- Now go to /etc/apt/apt.conf.d/50unattended-upgrades and uncomment the following lines. These will help to keep the system clean, minimize disk usage and make upgrades less disruptive.

Unattended-Upgrade::AutoFixInterruptedDpkg "true";
Unattended-Upgrade::Remove-Unused-Dependencies "true";

Unattended-Upgrade::Automatic-Reboot "true";
Unattended-Upgrade::Automatic-Reboot-Time "03:00";

```
// This option allows you to control if on a unclean dpkg exit
// unattended-upgrades will automatically run
//   dpkg --force-confold --configure -a
// The default is true, to ensure updates keep getting installed
Unattended-Upgrade::AutoFixInterruptedDpkg "true";
```

```
// Do automatic removal of unused packages after the upgrade
// (equivalent to apt-get autoremove)
Unattended-Upgrade::Remove-Unused-Dependencies "false";
```

```
// Automatically reboot *WITHOUT CONFIRMATION* if
//  the file /var/run/reboot-required is found after the upgrade
Unattended-Upgrade::Automatic-Reboot "true";
```

```
// If automatic reboot is enabled and needed, reboot at the specific
// time instead of immediately
//  Default: "now"
Unattended-Upgrade::Automatic-Reboot-Time "02:00";
```

- To test the above configuration, just run: "**sudo unattended-upgrade -d".** This simulates a full security upgrade and prints what would happen.

b. Container Update Procedure

To demonstrate automated security updates for container-based environments, I deployed Docker and ran a test nginx container. I then installed Watchtower, a lightweight tool that checks running containers and updates them if a new image version is available. Watchtower was configured to check every 60 seconds for this prototype. After pulling a newer nginx image, Watchtower successfully restarted the container with the updated version, proving the functionality.

Commands:

- Install Docker:
  sudo apt update
  sudo apt install docker.io
  sudo systemctl enable docker
  sudo systemctl start docker

- Deploy nginx as a test container: **sudo docker run -d --name test-nginx nginx**

- Install Watchtower:
  **sudo docker run -d **
  **--name watchtower **
  **--restart always **
  **-v /var/run/docker.sock:/var/run/docker.sock **
  **containrrr/watchtower **
  --cleanup --interval 60

```
tedi1@tedi1server:/$ sudo docker run -d --name test-nginx nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
b16f1b166780: Pull complete
c9a20772aff4: Pull complete
5c242ffc14bb: Pull complete
bf2e7af999d2: Pull complete
5cbad9890292: Pull complete
4cf85f4d417b: Pull complete
99f78d9a3fb1: Pull complete
Digest: sha256:fb39280b7b9eba5727c884a3c7810002e69e8f961cc373b89c92f14961d903a0
Status: Downloaded newer image for nginx:latest
2a7893a3ca2ef5067d5145e3b235a62caa3f9e180847192e6c11afa08205445d
tedi1@tedi1server:/$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS     NAMES
2a7893a3ca2e   nginx     "/docker-entrypoint.…"   7 seconds ago   Up 6 seconds   80/tcp    test-nginx
tedi1@tedi1server:/$ sudo docker run -d --name watchtower --restart always -v /var/run/docker.sock:/var
/run/docker.sock containrrr/watchtower --cleanup --interval 60
```

- Simulate an update: **sudo docker pull nginx.**
- If we check the logs using "**sudo docker logs watchtower"**, we will see that Watchtower is active, has detected 2 running containers and scanned them successfully. In this case no updates were made, because nginx image is already up to date.

```
tedi1@tedi1server:/$ sudo docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:fb39280b7b9eba5727c884a3c7810002e69e8f961cc373b89c92f14961d903a0
Status: Image is up to date for nginx:latest
docker.io/library/nginx:latest
tedi1@tedi1server:/$ sudo docker logs watchtower
time="2025-06-02T10:24:51Z" level=info msg="Watchtower 1.7.1"
time="2025-06-02T10:24:51Z" level=info msg="Using no notifications"
time="2025-06-02T10:24:51Z" level=info msg="Checking all containers (except explicitly disabled with la
bel)"
time="2025-06-02T10:24:51Z" level=info msg="Scheduling first run: 2025-06-02 10:25:51 +0000 UTC"
time="2025-06-02T10:24:51Z" level=info msg="Note that the first check will be performed in 59 seconds"
time="2025-06-02T10:25:52Z" level=info msg="Session done" Failed=0 Scanned=2 Updated=0 notify=no
tedi1@tedi1server:/$
```

1.4 Security Frameworks

For the security frameworks part of this project, I decided to go for AppArmor, which is a Mandatory Access Control (MAC) framework for Linux that restricts the capabilities of programs based on predefined security profiles. Unlike traditional discretionary access control (DAC) where users and programs can set file permissions, AppArmor enforces stricter rules, controlling what files, system resources, or capabilities each program can access — even if run by a privileged user.

In the context of a secure Ubuntu server, AppArmor helps prevent exploitation of vulnerabilities, in my case, exploitation of web server and databases which will be implemented in the next section.

For this part, I have been working based on both CIS Benchmark for Ubuntu LTS 22.04 and the book Mastering Linux Security and Hardening, along with research done by myself.

Commands:

a. AppArmor usually comes installed in Ubuntu systems and is already enabled. To check this, you should run "sudo aa-status". If AppArmor is already active, this command will return the loaded profiles, enforced and complain mode and unconfined processes.

```
tedi1@tedi1server:/$ sudo aa-status
[sudo] password for tedi1:
apparmor module is loaded.
150 profiles are loaded.
52 profiles are in enforce mode.
```

```
3 processes have profiles defined.
3 processes are in enforce mode.
   /usr/sbin/cups-browsed (35289)
   /usr/sbin/cupsd (35287)
   /usr/sbin/rsyslogd (1090) rsyslogd
0 processes are in complain mode.
0 processes are in prompt mode.
0 processes are in kill mode.
0 processes are unconfined but have a profile defined.
0 processes are in mixed mode.
```

b. I checked whether AppArmor was enabled in the bootlader configration. AppArmor must be enabled at boot time in the bootloader configuration to ensure that the controls it provides are not overridden. This can be done using: **grep "^\s*linux" /boot/grub/grub.cfg | grep -v "apparmor=1"** In my case, the above command return the following as shown in the screenshot, which means that AppArmor is active, but it was not explicitly enabled at boot time. Therefore I edited the file **/etc/default/grub** by adding the following line: **GRUB_CMDLINE_LINUX_DEFAULT="apparmor=1 security=apparmor"**

```
tedi1@tedi1server:/$ grep "^\s*linux" /boot/grub/grub.cfg | grep -v "apparmor=1"
grep: /boot/grub/grub.cfg: Permission denied
tedi1@tedi1server:/$ sudo grep "^\s*linux" /boot/grub/grub.cfg | grep -v "apparmor=1"
[sudo] password for tedi1:
        linux   /vmlinuz-6.8.0-60-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro
                linux   /vmlinuz-6.8.0-60-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro
                linux   /vmlinuz-6.8.0-60-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro recovery nomodeset
                linux   /vmlinuz-6.8.0-59-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro
                linux   /vmlinuz-6.8.0-59-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro recovery nomodeset
tedi1@tedi1server:/$
```

```
  GNU nano 7.2                                          /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=0
GRUB_DISTRIBUTOR=`( . /etc/os-release; echo ${NAME:-Ubuntu} ) 2>/dev/null || echo Ubuntu`
GRUB_CMDLINE_LINUX_DEFAULT=""
GRUB_CMDLINE_LINUX=""
```

```
GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=0
GRUB_DISTRIBUTOR=`( . /etc/os-release; echo ${NAME:-Ubuntu} ) 2>/dev/null || echo Ubuntu`
GRUB_CMDLINE_LINUX_DEFAULT="apparmor=1 security=apparmor"
GRUB_CMDLINE_LINUX=""
```

Afterwards, I ran **“sudo update-grub”** and **“sudo reboot”**. When the reboot was finished, the changes were applied and AppArmor is enabled even at boot time.

```
tedi1@tedi1server:/$ sudo update-grub
Sourcing file `/etc/default/grub'
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.8.0-60-generic
Found initrd image: /boot/initrd.img-6.8.0-60-generic
Found linux image: /boot/vmlinuz-6.8.0-59-generic
Found initrd image: /boot/initrd.img-6.8.0-59-generic
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
Adding boot menu entry for UEFI Firmware Settings ...
done
```

```
tedi1@tedi1server:/$ sudo grep "^\s*linux" /boot/grub/grub.cfg | grep -v "apparmor=1"
        linux   /vmlinuz-6.8.0-60-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro recovery nomodeset
        linux   /vmlinuz-6.8.0-59-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro recovery nomodeset
```

c. In order to see all available profiles, you should go to the directory **/etc/apparmor.d** and list all the available profiles. In my case, because I will be setting up a web server + database, I have to install the extra profiles along with apparmor-utils.

- **sudo apt install apparmor-profiles apparmor-profiles-extra**
- **sudo apt install apparmor-utils**
- In order to see the profiles, we can use “grep”

```
tedi1@tedi1server:/$ ls /etc/apparmor.d/ | grep apache
apache2.d
usr.sbin.apache2
tedi1@tedi1server:/$ ls /etc/apparmor.d/ | grep mysql
usr.sbin.mysqld
tedi1@tedi1server:/$
```

- Since I will be using both “apache2” and “mysql” and both of them are currently downloaded, I enforced both profiles. Additionally, I reviewed the current profiles in complain mode and decided to disable them since they were not necessary.

```
tedi1@tedi1server:/$ sudo aa-enforce /etc/apparmor.d/usr.sbin.apache2
[sudo] password for tedi1:
Setting /etc/apparmor.d/usr.sbin.apache2 to enforce mode.
tedi1@tedi1server:/$ sudo aa-enforce /etc/apparmor.d/usr.sbin.mysqld
Setting /etc/apparmor.d/usr.sbin.mysqld to enforce mode.
tedi1@tedi1server:/$
```

```
8 profiles are in complain mode.
   /usr/bin/irssi
   /usr/sbin/sssd
   libreoffice-oosplash
   libreoffice-soffice
   transmission-cli
   transmission-daemon
   transmission-gtk
   transmission-qt
```

1.5 Hardening Scenario

For the hardening scenario, I decided to go for a web server application with a database, more specifically **Drupal**.

a. Download the necessary packages for Drupal:
sudo apt install apache2 mysql-server php php-mysql libapache2-mod-php php-xml php-gd php-mbstring php-curl php-zip php-json php-dom php-opcache unzip curl gnupg2 -y

b. Launch secure installation of mysql: **sudo mysql_secure_installation**
Choose:

- **Set a strong root password**
- **Remove anonymous users**
- **Disallow root remote login**
- **Remove test database**
- **Reload privileges**

```
tedi1@tedi1server:~$ sudo mysql_secure_installation

Securing the MySQL server deployment.

Connecting to MySQL using a blank password.

VALIDATE PASSWORD COMPONENT can be used to test passwords
and improve security. It checks the strength of password
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD component?

Press y|Y for Yes, any other key for No: y

There are three levels of password validation policy:

LOW    Length >= 8
MEDIUM Length >= 8, numeric, mixed case, and special characters
STRONG Length >= 8, numeric, mixed case, special characters and dictionary
            file

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG:
```

c. Lock it to localhost

- Go to **/etc/mysql/mysql.conf.d/mysqld.cnf**
- Set **bind-address: 127.0.0.1**
- Restart: **sudo systemctl restart mysql**

```
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 127.0.0.1
mysqlx-bind-address     = 127.0.0.1
```

d. Create a Drupal database and user:

- sudo mysql -u root -p
- CREATE DATABASE drupal CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
  CREATE USER 'drupaluser'@'localhost' IDENTIFIED BY 'StrongDrupalPass123!';
  GRANT ALL PRIVILEGES ON drupal.* TO 'drupaluser'@'localhost';
  FLUSH PRIVILEGES;
  EXIT;

```
tedi1@tedi1server:~$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.42-0ubuntu0.24.04.1 (Ubuntu)
```

```
mysql> CREATE DATABASE drupal CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
Query OK, 1 row affected (0.00 sec)

mysql> CREATE USER 'drupaluser'@'localhost' IDENTIFIED BY 'StrongDrupalPass123!'
;
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT ALL PRIVILEGES ON drupal.* TO 'drupaluser'@'localhost';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.01 sec)
```

e. Download and install Drupal:
cd /tmp
curl -O https://ftp.drupal.org/files/projects/drupal-10.2.5.tar.gz
tar -xvzf drupal-10.2.5.tar.gz
sudo mv drupal-10.2.5 /var/www/html/drupal

```
tedi1@tedi1server:/tmp$ curl -O https://ftp.drupal.org/files/projects/drupal-10.2.5.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:06 --:--:--     0
100 18.0M  100 18.0M    0     0   755k      0  0:00:24  0:00:24 --:--:-- 4222k
```

f. Set the permissions

```
tedi1@tedi1server:/tmp$ sudo chown -R www-data:www-data /var/www/html/drupal
tedi1@tedi1server:/tmp$ sudo chmod -R 755 /var/www/html/drupal
```

g. Configure Apache for Drupal

- Create a new config: sudo nano /etc/apache2/sites-available/drupal.conf
- Add the following:

<VirtualHost *:80>
ServerAdmin admin@localhost
DocumentRoot /var/www/html/drupal
<Directory /var/www/html/drupal>
AllowOverride All
Require all granted
</Directory>
ErrorLog ${APACHE_LOG_DIR}/drupal_error.log
CustomLog ${APACHE_LOG_DIR}/drupal_access.log combined
</VirtualHost>

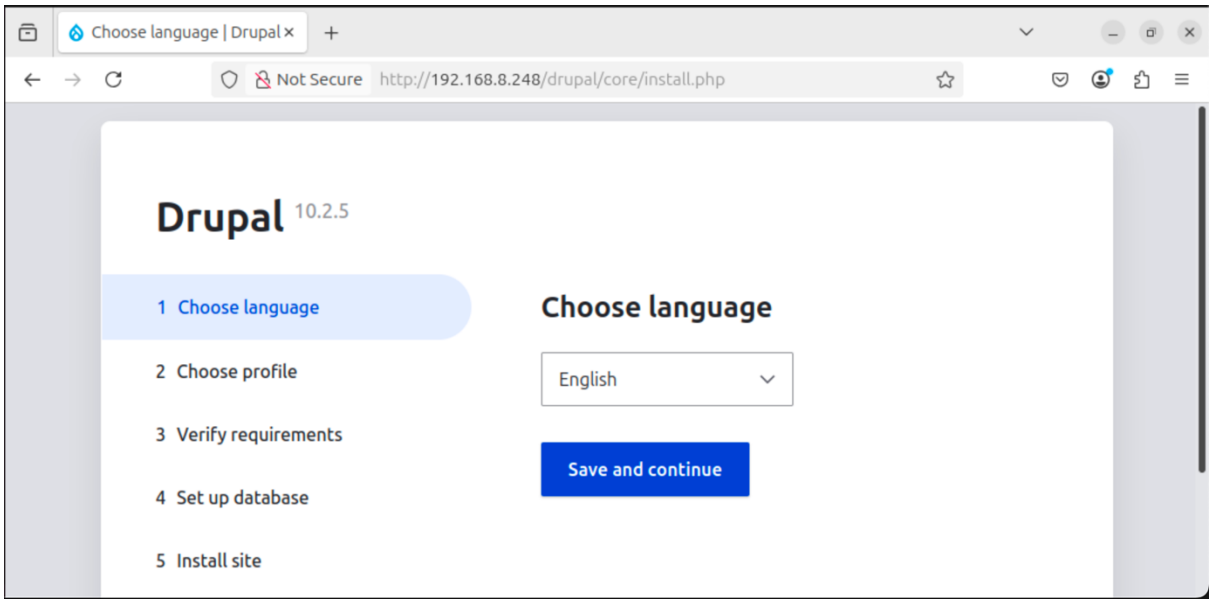
```
tedi1@tedi1server:/$ sudo nano /etc/apache2/sites-available/drupal.conf
tedi1@tedi1server:/$ sudo a2ensite drupal.conf
Enabling site drupal.
To activate the new configuration, you need to run:
  systemctl reload apache2
tedi1@tedi1server:/$ sudo a2enmod rewrite
Enabling module rewrite.
To activate the new configuration, you need to run:
  systemctl restart apache2
```

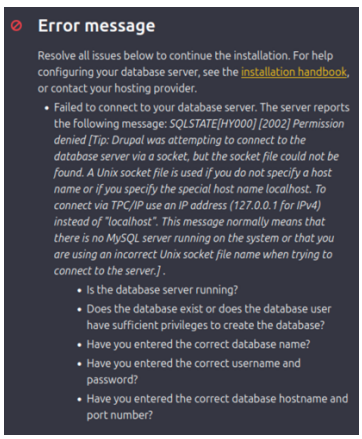
h. Check the status of the apache2

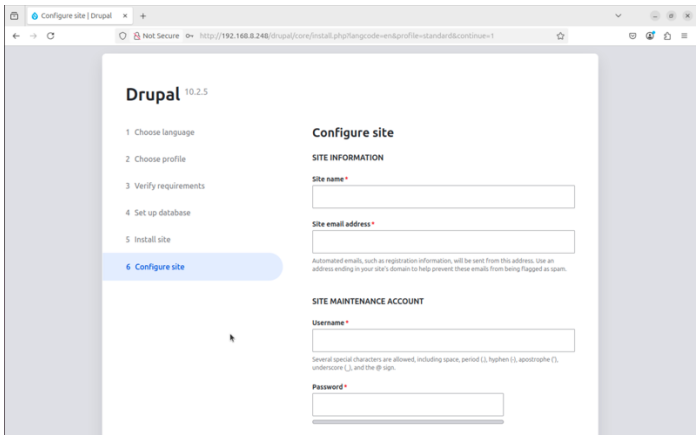
```
tedi1@tedi1server:/$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-06-02 21:26:47 UTC; 32s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 19150 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 19155 (apache2)
      Tasks: 6 (limit: 4542)
     Memory: 13.5M (peak: 14.5M)
        CPU: 34ms
     CGroup: /system.slice/apache2.service
             ├─19155 /usr/sbin/apache2 -k start
             ├─19157 /usr/sbin/apache2 -k start
             ├─19158 /usr/sbin/apache2 -k start
             ├─19159 /usr/sbin/apache2 -k start
             ├─19160 /usr/sbin/apache2 -k start
             └─19161 /usr/sbin/apache2 -k start

Jun 02 21:26:47 tedi1server systemd[1]: Starting apache2.service - The Apache HTTP Server...
Jun 02 21:26:47 tedi1server systemd[1]: Started apache2.service - The Apache HTTP Server.
tedi1@tedi1server:/$
```

i. Now try to open it on web: **http://192.168.8.248/drupal.** As we can see, it workes properly and we should set it up properly. I encountered an error during the database setup, because localhost should be changed to 127.0.0.1, so that it connect properly.







j. Once we finish this procedure, we can see that the site is fully functional.

k. Following this, I made some security changes to the site:
   - Installed the following: Update Manager, Automated Logout, Security Kit, Captcha, Password Policy, Two-Factor Auth
   - I made sure ".htaccess" in /var/www/html/drupal/ was present and not deleted. Also I add the following line to prevent PHP execution:
     <FilesMatch "\.php$">
     Deny from all
     </FilesMatch>
   - I changed the permission on setting.php to make it more secure:
     sudo chmod 444 /var/www/html/drupal/sites/default/settings.php
   - Enforced HTTPS. In **/etc/apache2/sites-available/000-default.conf,** added the following**:**
     <VirtualHost *:80>
     ServerName YOUR_SERVER_IP
     Redirect "/" "https://YOUR_SERVER_IP/"
     </VirtualHost>
   - Then I set HSTS Header (HTTP Strict Transport Security). Inside /etc/apache2/sites-available/drupal-ssl.conf, I added the following:
     Header always set Strict-Transport-Security "max-age=63072000; includeSubDomains; preload"
   - Enabled and configured Cron jobs. Enter the command "sudo crontab -e" and then:
     0 */6 * * * wget -O - -q -t 1 https://192.168.8.248/cron/UNIQUE_CRON_KEY

1.6 Extract Log Data

For this project, I used rsyslog for the logging of data. The log data will be transferred using TCP on port 514 to another VM, where a log server is set up. The transferred data is converted to JSON and saved in MongoDB.

Rsyslog Setup

a. Install rsyslog: sudo apt install rsyslog
b. Enable TCP reception in rsyslog:
   - Go to /etc/rsyslog.conf
   - Uncomment these two lines:
     module(load="imtcp")
     input(type="imtcp" port="514")
     This will load the TCP module and and open port 514 for incoming logs

c. Also I updated the firewall rules to allow port 514/tcp: sudo ufw allow 514/tcp

```
# provides TCP syslog reception
module(load="imtcp")
input(type="imtcp" port="514")
```

d. In order to export the logs to a safe location, I created a new virtual machine, where I set up a server to store the logs. For the second VM, I also used the Ubuntu 24.04.2 LTS, but with a minimised installation to avoid the unnecessary packages and lower the attack surface. For the user of this VM, I repeated the same steps as above regarding the password criteria in pwquality and also set up Google Authenticator time-based code.

```
# provides TCP syslog reception
module(load="imtcp")
input(type="imtcp" port="514")
```

e. In the second VM, I also disabled DHCP and set a static IP address to it (192.168.8.212/24) and applied the changes with **"sudo netplan apply"**

```
 GNU nano 7.2                                  /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    ens160:
      dhcp4: false
      addresses: [192.168.8.212/24]
      gateway4: 192.168.8.1
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
```

f. Now go back to the first VM (tedi1server), where the rsyslog file is created.
- Create a new file: **sudo nano /etc/rsyslog.d/60-remote.conf**
- Add the following line: ***.* @@192.168.8.212:514,** where:
  . stands for: forward all logs
  @@ stands for: use TCP (reliable)
  192.168.8.212: is the static IP of the log server
- Restart rsyslog: **sudo systemctl restart rsyslog**

g. To test the rsyslog, try a logger text in the first VM (tedi1server) and then check the saved logs in the second VM:
- First VM: **logger "Test log message from tedi1server"**
- Second VM: run the command "**sudo tail -n 50 /var/log/syslog"**. You will see that all the logs of the first VM are being passed to the second VM.

```
tedi1@tedi1server:~$ logger "Test log message from tedi1server"
tedi1@tedi1server:~$
```

```
2025-06-02T18:34:50+00:00 tedi1server tedi1: Test log message from tedi1server
2025-06-02T18:34:54+00:00 tedi1server dbus-daemon[2575]: [session uid=1000 pid=2575]
```

l. Now comes the database setup. For the database, I used MongoDB, but since MongoDB is not available for Ubuntu LTS 24, I installed it inside a Docker container
- Install Docker:
  sudo apt update
  sudo apt install docker.io -y
  sudo systemctl enable docker
  sudo systemctl start docker

- Run mongodb on port 27017 of the log server: **sudo docker run -d --name mongodb -p 27017:27017 mongo:6**

- Install rsyslog-mongodb on the log server: **sudo apt install rsyslog-mongodb -y**

```
0 upgraded, 6 newly installed, 0 to remove and 54 not upgraded.
Need to get 1859 kB of archives.
After this operation, 4781 kB of additional disk space will be used.
Get:1 http://ports.ubuntu.com/ubuntu-ports noble/universe arm64 libbson-1.0-0t64 arm64 1.26.0-1.1ubuntu2 [83.1 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports noble/universe arm64 libmongocrypt0 arm64 1.8.4-1build3 [1307 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports noble/main arm64 libsnappy1v5 arm64 1.1.10-1build1 [28.6 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports noble/universe arm64 libutf8proc3 arm64 2.9.0-1build1 [71.1 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports noble/universe arm64 libmongoc-1.0-0t64 arm64 1.26.0-1.1ubuntu2 [356 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports noble-updates/universe arm64 rsyslog-mongodb arm64 8.2312.0-3ubuntu9.1 [12.1 kB]
Fetched 1859 kB in 1s (1307 kB/s)
```

- Now create a new configuration file: **sudo nano /etc/rsyslog.d/30-mongo-json.conf.** It will have the following content:
  module(load="ommongodb")

  template(name="json-template" type="list") {
  constant(value="{")
  constant(value="\"timestamp\":\"") property(name="timereported"
  dateFormat="rfc3339")
  constant(value="\",\"host\":\"") property(name="hostname")
  constant(value="\",\"severity\":\"") property(name="syslogseverity-text")
  constant(value="\",\"facility\":\"") property(name="syslogfacility-text")
  constant(value="\",\"tag\":\"") property(name="syslogtag")
  constant(value="\",\"message\":\"") property(name="msg" format="json")
  constant(value="\"}")
  }

  action(
  type="ommongodb"
  server="localhost"
  db="logs"
  collection="syslog"
  template="json-template"
)

```
tedi2@tedilogserver:~$ sudo cat /etc/rsyslog.d/
cat: /etc/rsyslog.d/: Is a directory
tedi2@tedilogserver:~$ sudo cat /etc/rsyslog.d/30-mongo-json.conf
module(load="ommongodb")

template(name="json-template" type="list") {
  constant(value="{")
  constant(value="\"timestamp\":\"")     property(name="timereported" dateFormat="rfc3339")
  constant(value="\",\"host\":\"")       property(name="hostname")
  constant(value="\",\"severity\":\"")   property(name="syslogseverity-text")
  constant(value="\",\"facility\":\"")   property(name="syslogfacility-text")
  constant(value="\",\"tag\":\"")        property(name="syslogtag")
  constant(value="\",\"message\":\"")    property(name="msg" format="json")
  constant(value="\"}")
}

action(
  type="ommongodb"
  server="localhost"
  db="logs"
  collection="syslog"
  template="json-template"
)

tedi2@tedilogserver:~$
```

- Restart syslog**: sudo systemctl restart rsyslog**
- Go to the docker container: **sudo docker exec -it mongodb mongosh**
- Run inside mongosh:
  use logs
  db.syslog.find().pretty()
  Now you can see all the logs that are stored in the MongoDB in a JSON format.

```
{
  _id: ObjectId('683e02162c52feea3100d8a6'),
  timereported: '2025-06-02T19:57:10.631476+00:00',
  hostname: 'tedilogserver',
  'syslogseverity-text': 'info',
  'syslogfacility-text': 'daemon',
  syslogtag: 'systemd[1]:',
  msg: ' Starting rsyslog.service - System Logging Service...'
},
{
  _id: ObjectId('683e02162c52feea3100d8a7'),
  timereported: '2025-06-02T19:57:10.687004+00:00',
  hostname: 'tedilogserver',
  'syslogseverity-text': 'info',
  'syslogfacility-text': 'syslog',
  syslogtag: 'rsyslogd:',
  msg: " imuxsock: Acquired UNIX socket '\\/run\\/systemd\\/journal\\/syslog' (fd 3) from systemd.  [v8.2312.0]"
},
{
  _id: ObjectId('683e02162c52feea3100d8a8'),
  timereported: '2025-06-02T19:57:10.687069+00:00',
  hostname: 'tedilogserver',
  'syslogseverity-text': 'info',
  'syslogfacility-text': 'syslog',
  syslogtag: 'rsyslogd:',
  msg: " rsyslogd's groupid changed to 104"
},
```