**TABLE OF CONTENTS:**

* **AUDITING DOCKER SECURITY**
  + Security Audit
  + Docker Bench for Security
  + Auditing Docker Security with Docker Bench for Security
* **SECURING THE DOCKER HOST**
  + Host Security
  + Security Auditing with Lynis
    - Installing and Running Lynis
    - Creating User Account
    - Setting up SUDO Access
    - Adding User to the Docker Group
    - Disabling root Logins
    - Securing SSH
    - Setting Up key-Based Authentication with SSH
    - Disable Password Authentication with SSH
  + Setting Up Audit Rules for Docker Artifacts
    - The Linux Audit Framework
    - Installing Auditd
    - Creating Audit Rules for Docker Artifacts
* **SECURING THE DOCKER DAEMON**
  + Implementing TLS Encryption
    - Generating TLS Certificates
    - Docker Daemon Configuration
  + Implementing User Namespaces
  + Correcting Docker Daemon Configuration Warnings
  + Running Docker Bench for Security after Securing the Docker Daemon
* **CONTAINER SECURITY BEST PRACTICES**
  + Using an Unprivileged User
  + Disabling the root User
  + Preventing Privilege Escalation Attacks
  + Limiting Container Capabilities
  + Filesystem Permissions and Access
  + Disabling Inter-Container Communication
* **CGROUPS**
* **ACCESS CONTROL WITH APPARMOR**
* **LIMITING CONTAINER SYSTEM CALLS WITH SECCOMP**

**AUDITING DOCKER SECURITY**

The first step in the process of securing a system is to perform a security audit. An audit establishes a baseline of the security of a system. This initial baseline will be used to guide us in accordance with what needs to be secured. Before we get started with the security auditing process, we need to understand what a security audit is and why it is important in securing a system.

**WHAT IS A SECURITY AUDIT?**

A security audit is a systematic evaluation of the security and configuration of a particular information system. Security audits are used to measure the security performance of a system against a list of checks, best practices, and standards. In the case of Docker, we will be using the CIS Docker Benchmark, which is a consensus driven security guideline for the Docker platform. The CIS Docker Benchmark provides us with a solid set of guidelines and checks that can be used to test the security of the Docker platform and establish a baseline security level. More information about the CIS Docker Benchmark can be found here: **https://www.cisecurity.org/benchmark/docker**

**DOCKER BENCH FOR SECURITY**

Docker Bench for Security is an open-source Bash script that checks for various common security best practices of deploying Docker in production environments. The tests are all automated and are based on the CIS Docker Benchmark. More information about Docker Bench for Security can be found on GitHub: [**https://github.com/docker/docker-bench-security**](https://github.com/docker/docker-bench-security)**.**

**AUDITING DOCKER SECURITY WITH DOCKER BENCH FOR SECURITY**

1) Clone the repository of docker-bench-security using the below command:

git clone <https://github.com/docker/docker-bench-security.git>

2) After cloning the repository, navigate to the directory you just cloned using the below command:

cd docker-bench-security

3) Run the below bash script command to perform docker security audit:

sudo ./docker-bench-security.sh

4)When the script is executed, it will perform all the necessary security checks. Once completed, it will provide you with a baseline security score as highlighted in the image below.

**Graphical user interface, application

Description automatically generated**

The initial baseline security score will be valued at zero, indicating that all checks failed. In this case, we can identify what needs to be secured by analysing the results produced by the script, as highlighted in the image below.

A picture containing table

Description automatically generated

Each check performed by the script is numbered and is flagged with the corresponding color code based on whether the check was successful:

WARN: The corresponding check failed, indicating its need to be secured.

INFO: The check was run with no warning.

PASS: The corresponding check was run successfully.

The script also provides a list of recommendations regarding what components need to be secured for every check. See the image above

The script also sorts the results based on the following categories:

* Host Configuration
* General Configuration
* Docker Daemon Configuration
* Docker Swarm Configuration

This categorization of checks is very useful as it distinguishes the security of components from others, therefore streamlining the process.

The first component that we need to secure based on the results is the Docker host. Let’s take a look at how to secure the Docker host and implement the security practices recommended by the Docker Bench for Security tool.

**SECURING THE DOCKER HOST**

Given the fact that Docker containers utilize the host OS kernel, the Docker platform and its containers are only as secure as the host operating system.

our host OS is running Linux, but similar principles should be followed for other operating systems.

**HOST SECURITY**

The security of the host kernel and operating system will have a direct correlation to the security of your containers, given the fact that the containers utilize the host kernel. It is therefore vitaly important to keep your host secure. The following guidelines outline various security best practices you should consider when securing your Docker host:

1. Consider the use of minimal Linux distributions that offer a much smaller attack surface.
2. Secure and harden your host OS.
3. Ensure your host OS is kept up to date.
4. Ensure your kernel is up to date.
5. Ensure you have the latest version of Docker running.
6. Add your host and containers to a robust vulnerability management plan and constantly scan your host and containers for vulnerabilities.
7. Only run the services you need to run.
8. Keep up with the latest vulnerability news for the Linux kernel and the Docker platform.

The process of securing the host OS is multi-faceted and leverages multiple security audit tools in order to establish a baseline security level. This process will result in a Docker host that satisfies the CIS Docker Benchmark.

We will address securing the Docker host in two parts:

* We will run an operating system security audit tool called Lynis. This will help us secure and harden the host OS. We will implement the recommendations made by Lynis.
* We harden the host OS, we will return to the Docker Bench for Security to enable and set up auditing for our Docker components and artifacts.

**SECURITY AUDITING WITH LYNIS**

Lynis is an extensible security audit tool for computer systems running Linux, FreeBSD, macOS, OpenBSD, Solaris, and other Unix derivatives. It assists system administrators and security professionals with scanning a system and its security defences, with the final goal being system hardening.

**INSTALLING LYNIS**

Run the following command to install LYNIS:

sudo apt-get install lynis

Prior to Scanning, we need to ensure that LYNIS is up to date. Type the following command to check if latest version of LYNIS is installed.

sudo lynis update info

**Text

Description automatically generated**

**RUNNING LYNIS**

Perform a system audit with Lynis, run the following command: sudo lynis audit system

Lynis will output a lot of information that will also be stored under the /var/log/lynis.log file for easier access. The summary of the system audit will reveal important information about your system’s security posture and various security misconfigurations and vulnerabilities

Lynis will also generate output on how these vulnerabilities and misconfigurations can be fixed or tweaked.

**Text

Description automatically generated**

The output also contains a hardening index score that is rated out of 100. This is used to give you a trackable tangible score of your system’s current security posture. Lynis will also display any potential warnings that indicate a severe security vulnerability or misconfiguration that needs to be fixed or patched

To increase our hardening index score, Lynis provides us with helpful suggestions that detail the various security configurations we need to make.

We can now follow the recommendations provided by Lynis to secure and harden our Docker host.

**CREATING A USER ACCOUNT**

To Secure a Host OS:

* The first step is to add and configure the necessary user accounts on the system.
* We then need to set up the various groups that will be used to assign permissions to specific users with specific roles.
* After, we will begin specifying file permissions and assigning ownership of specific files and directories. This will help us set up a system of accountability and defence in depth.

Linux has multi-user support and, as a result, multiple users can access the system simultaneously. This can be seen as both an advantage and disadvantage from a security perspective in that multiple accounts offer multiple access vectors for attackers and therefore increase the overall risk of the server. To counter this concern, we must ensure that user accounts are set up and sorted accordingly in terms of their privileges and roles.

Example: Having multiple users on a Linux server with root privileges is extremely dangerous as an attacker will only need to compromise one account to get root access on the system. We can easily solve this issue by segregating permissions for users based on their roles.

Creating a user account on Linux can be done by following the steps outlined below:

* The useradd command creates users on your system and has this general syntax:

useradd <arguments> username

* The arguments in the above command can be:

|  |  |
| --- | --- |
| Argument | Function |
| -c | A text string that is used to include comments about the account, like the user’s first and last name. |
| -m | When included, this option tells the useradd command to create a home directory for the new user. |
| -s | Used to specify the user’s login shell (e.g. /bin/bash, /bin/zsh, etc). |

* Use the below command to create a user account:

sudo useradd -c “First Name Last Name” -m -s /bin/bash <username>

example:



* We now need to specify the password for the user account. We can do this with the following command:

sudo passwd <username>

example:

Text

Description automatically generated

**SETTING UP SUDO ACCESS**

When setting up access on a Linux server, some users may require sudo access to perform administrative tasks like updating packages and installing software. By default, users do not have sudo access, which means they are unable to perform these administrative tasks.

Giving a user sudo access involves adding the user to a sudo-enabled group. By default, this group is just called sudo on Debian-based systems, and on Fedora and RedHat-based systems this group is called wheel. One way we can add the user we have just created to the sudo group by running the following command:

usermod -aG sudo <username>

example:



**ADDING USER TO THE DOCKER GROUP**

Docker implements access control for the Docker daemon through a Linux group with specific permissions. Members of this group will have the privileges required to interact with the Docker daemon. As a result, only authorized users that require access should be added to this group.

We can add our custom user to this group by running the following command:

usermod -aG docker <username>

example:

****

**DISABLING ROOT LOGINS**

The first step in setting up local authentication security is to disable root logins. Following this step prevents any authorized or unauthorized user from gaining access to the root user account and consequently the server because the root user has complete power over the system.

The root user’s privileges can be abused to run any commands provided (malicious or otherwise), including modifying the passwords of other users on the system, consequently locking them out. Common Linux security practices recommend disabling root logins and creating a separate administrative account, which can be assigned sudo privileges to run certain commands with root privileges. Following this step will help mitigate the threats to the root account and will reduce the overall attack surface of the host.

We can disable root logins in a few different ways. The first method of disabling root logins is by changing the default shell of the root user from /bin/bash or /bin/sh to /usr/sbin/nologin. This can be done by using the chsh (Change Shell) utility on Linux:

* Run the following command:

sudo chsh root

* After running the command, we will be prompted to enter the absolute path of the shell we want to switch to. Specify **/usr/sbin/nologin** as the shell at the prompt.

Text

Description automatically generated

* After you have entered the absolute path to the nologin shell, we can try logging in to the root account. When attempting to log in, the message appears, and we are unable to log into the root account.



* These changes will prevent unauthorized users from using the root account, because we have not specified a valid shell. However, users with sudo privileges will still be able to run all administrative commands unless the privileges are constrained to certain commands.

**Note:** Aside from using the chsh utility, another way to update the user’s shell is to modify the /etc/passwd file

Now that we have disabled root user logins, we will be using the custom user account that we have created going forward. The next step in authentication security involves securing the remote access protocol, which in most cases will be SSH.

**SECURING SSH**

If your system did not have root password logins disabled, then any attacker could attempt to gain root access by performing password brute-force attacks on the SSH protocol. So, it’s important to disable root login via SSH as well.

It’s also important to do this even if you do have root password logins disabled, because it adds an extra layer of security.

* We can disable root login via SSH by modifying the OpenSSH server configuration file found in /etc/ssh/sshd\_config.
* After opening the file with a text editor like nano or vim, we will be greeted with extensive configuration options that we can use to modify how the SSH server will function

.

Text

Description automatically generated

* To disable root login with SSH, we need to change the PermitRootLogin configuration from yes to no. The authentication configurations can be found under the #Authentication section. Ensure that you also uncomment the configuration to activate it by removing the # symbol at the beginning of the PermitRootLogin line.

**Text

Description automatically generated**

* As you can see in the image above, we have set the option from yes to no. This will prevent users from authenticating via SSH as the root user.
* 5. After saving the file, we now need to restart the SSH service. This can be done by running the following command:

sudo systemctl restart sshd

* After restarting the SSH daemon on the server, we can try logging in to the root account remotely via SSH. As you can see in the image below we get a Permission Denied error even after entering the correct root password. This confirms that we have successfully disabled root logins via SSH

**Text

Description automatically generated**

**SETTING UP KEY-BASED AUTHENTICATION WITH SSH**

Key-based authentication utilizes asymmetric encryption to generate two keys that are used for the encryption and decryption of data. These two keys are called the public key and the private key, and together they are called a public-private key pair.

The public key is used to encrypt data and only the corresponding private key can decrypt the data. As a result, the private key must be kept private and secure, whereas the public key can be shared.

* SSH key pairs can be generated on the client by using the ssh-keygen utility. We can generate the key pair by running the following command:

ssh-keygen -t rsa

* This will generate the public and private RSA key pair, and you will be prompted to specify the directory to which you want to save the keys. You will also be prompted to specify a passphrase for the key pair. This is an additional level of security that you can use to secure your key pair.

**Text

Description automatically generated**

* The key pair will be generated and saved in your ~/.ssh/ directory. In this directory, you will find your public key with the .pub extension (e.g. id\_rsa.pub), and your private key with no file extension (e.g. id\_rsa).
* Your public key now needs to be uploaded to your server. We can do this with the ssh-copy-id utility:

ssh-copy-id <username@serverIP>

* We are now able to log in directly without entering a user password. Note that if you previously supplied a passphrase to the ssh-keygen utility, you will be prompted to enter that passphrase when logging in.

**DISABLE PASSWORD AUTHENTICATION WITH SSH**

We can now login with our private key. The next step is to disable password authentication completely, which will ensure that no user will be able to authenticate remotely with SSH without their respective key pair.

* This can be done by modifying the /etc/ssh/sshd\_config OpenSSH configuration file and setting the PasswordAuthentication option to no:

****

* After saving the new changes to the OpenSSH configuration file, restart the SSH daemon

sudo systemctl restart sshd

* The SSH server will restart with the new changes applied.

We have now secured the new user account and root account from unauthorized remote access. We are only able to login to the user account with the unique private key from the key pair we generated.

**SETTING UP AUDIT RULES FOR DOCKER ARTIFACTS**

During the initial Docker security audit, we performed with the Docker Bench for Security utility, we were able to identify several host configuration warnings that required us to set up audit rules for specific Docker artifacts. Examples of these artifacts include configuration files, binaries, and systemd service files

We can perform the Docker Bench for Security utility again. This time, we can limit our results to the host configuration to focus on just those checks.

This can be done by running the following command:

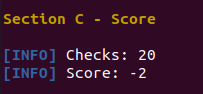
sudo ./docker-bench-security.sh -c host\_configuration

**Text

Description automatically generated**

The script should output a list of Docker artifacts that require audit rules. Before we can enable auditing of these artifacts.

File and object auditing allows us to log and analyse all the activity of an object. Auditing on Linux is facilitated through the Linux Audit Framework. In the context of auditing, an object is a system resource like a file, directory, application, or service. Docker requires us to have audit rules for core artifacts, like the Docker daemon, in order to ensure that all activity from these artifacts is logged for security purposes.



**THE LINUX AUDIT FRAMEWORK**

The Linux Audit Framework is used to set up and configure auditing policies for user-space processes like Docker.

The following diagram outlines the various components that make up the Linux Audit Framework and how they interact with each other:

**Diagram

Description automatically generated**

Here’s an overview of some of The Linux Audit Framework components:

* Auditd: The Audit daemon. This saves audit events to the audit log.
* Audit Log: Contains event logs from all configured audit rules.
* Auditctl: Client software that is used to manage and control the framework and is also used to create or delete audit rules
* Audit.rules: A configuration file that contains audit rules and is accessed by auditd when the service is restarted.

**INSTALLING AUDITD**

Enter the following command to install auditd:

sudo apt-get install auditd

**CREATING AUDIT RULES FOR DOCKER ARTIFACTS**

The host configuration security audit we performed with Docker Bench for Security provided a list of Docker artifacts that require auditing. We can create audit rules for these artifacts by running the following auditctl command:

sudo auditctl -w <PATH TO ARTIFACT> -k docker

|  |  |
| --- | --- |
| ARGUMENT | FUNCTION |
| -w | Used to specify the file or service to watch |
| -k | Used to specify the filter key, which is a short string of text. The same key can be applied to several different audit rules. By applying a key, you can group different rules together, which can be useful when analysing and searching through your audit logs. |

We need to create audit rules for all the artifacts listed in the audit results from the Docker Bench for Security utility

* In the previous Docker Bench for Security report, warnings like the following appeared:

****

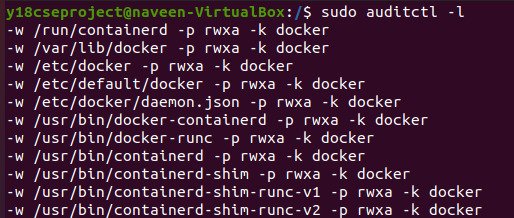
* For this warning, create a corresponding audit rule with a command like this:

sudo auditctl -w /etc/docker -k docker

* Create audit rules for each such warning. After creating the audit rules, we can list them by running the following command:

sudo auditctl -l

* This will list out all the created audit rules for the Docker artifacts. You should have similar rules to the ones highlighted in the image below:



* After creating the rules, we need to save them to the audit.rules file to make them permanent. This can be done by copying and pasting the audit rules from the output of the sudo auditctl -l command to the audit.rules file located in:

/etc/audit/rules.d/audit.rules

* After adding the rules to the audit.rules file, you will need to restart the auditd service. This can be done by running the following command:

sudo systemctl restart auditd

* After restarting auditd, we can re-run the Docker Bench for Security tool to confirm that the audit rules have been enabled and are active:

sudo ./docker-bench-security.sh -c host\_configuration

* As illustrated in the image below, the host configuration checks related to the auditing of Docker artifacts should all be successful:

A picture containing text

Description automatically generated

We should also have a new audit score that reflects the audit rules we have created as shown in the image below:

Text

Description automatically generated

**Now that we have been able to successfully secure our Docker host, we can begin the process of securing the Docker daemon**

**SECURING DOCKER DAEMON**

Now that we have a secure Docker host to work with, we can begin the process of securing the Docker daemon from the recommendations provided by the Docker Bench for Security tool.

The components that we need to implement are:

* TLS Encryption between the Docker client and Docker Daemon
* User Namespaces

**IMPLEMENTING TLS ENCRYPTION**

The communication between the Docker client and daemon can be performed locally through a UNIX domain socket or remotely using a TCP socket.

This communication is **not encrypted** by default and, as a result, an attacker can perform a **man in the middle (MITM) attack** andcan intercept the commands being sent remotely from the Docker client to the Docker daemon.

To mitigate this situation, we implement TLS Encryption for remote connections.

The process involves generating the TLS certificates for the server and the remote clients. We will begin by looking at how to generate the TLS certificates for both the Docker client and server, and then we will update the Docker daemon configuration to use the certificates.

**GENERATING TLS CERTIFICATES**

The process of generating TLS certificates manually can be slightly complicated. As a result, we will be using an automated Bash script to generate the certificates for us. This can be done by following the procedures outlined below:

* The first step in the process involves downloading the Bash script. This can be done by running the following commands:

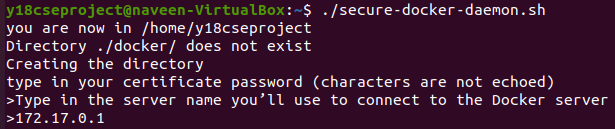
wget https://raw.githubusercontent.com/AlexisAhmed/ DockerSecurityEssentials/main/Docker-TLS-Authentication/secure- docker-daemon.sh

chmod u+x secure-docker-daemon.sh

* After downloading the script, we can execute it by running the following command:

./secure-docker-daemon.sh

The script will create a .docker/ directory in your user’s home directory as illustrated in the image below. This is where the certificates will be stored.

****

* The script will prompt you to enter the Docker server IP. After providing the IP address, the script will automatically create the client and server certificates in the .docker/ directory as highlighted in the image below:

**Text

Description automatically generated with medium confidence**

**DOCKER DAEMON CONFIGURATION**

After generating the TLS certificates, we now need to create a custom systemd configuration file for the Docker daemon. This configuration file will be used to enable TLS and specify the TLS certificates:

* + We can create the custom systemd file with a text editor and add the TLS configuration to it. Create the systemd file in your preferred text editor with a command like the following:

sudo mkdir /etc/systemd/system/docker.service.d/

sudo nano /etc/systemd/system/docker.service.d/override.conf

* + After creating and opening the file in your editor, add the following configuration to it. When pasting this snippet into your file, **be sure to replace the string with the username on your system**:

[Service]

ExecStart=

ExecStart=/usr/bin/dockerd -D -H unix:///var/run/docker.sock

--tlsverify --tlscert=/home/<user>/.docker/server-cert.pem

--tlscacert=/home/<user>/.docker/ca.pem --tlskey=/home/<user>/.

docker/server-key.pem -H tcp://0.0.0.0:2376

* + After adding the configuration to the file, we need to save it and restart the Docker service. This can be done by running the following command:

sudo systemctl restart docker

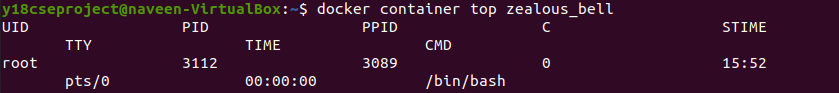
* + If the configuration file has been created and set up correctly, the Docker service should restart with no issues
  + You can now copy over the client TLS certificates to the remote Docker client for authentication.
  + More information regarding TLS authentication can be found here: https://docs.docker.com/engine/ security/protect-access/

Now that we have configured TLS encryption between the Docker client and daemon, we can move on to implementing user namespaces for containers.

**IMPLEMENTING USER NAMESPACES**

After generating the TLS certificates, we need to create a custom systemd configuration file for the Docker daemon. This configuration file will be used to enable TLS and specify the TLS certificates.

When we run a Docker container, the process is run from the default namespace. As a result, the process is run under the root user as highlighted in the image below:

****

This can be dangerous in the event of a container breakout. Because the process is being run as the root user, an attacker would be able to get root privileges for the host. As a result, **we need to run containers as an unprivileged user**.

We need to reconfigure the Docker daemon to use user namespaces. Docker generates a default dockremap user that you can use, or you can specify your own non-privileged user.

* We can implement user namespaces by adding the following option to the ExecStart line in the /etc/systemd/system/docker.service.d/override.conf file we created in the previous section:

--userns-remap=”default”

* The new configuration should be structured as follows:

[Service]

ExecStart=

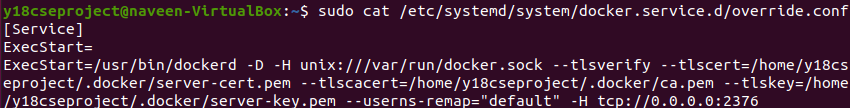
ExecStart=/usr/bin/dockerd -D -H unix:///var/run/docker.sock

--tlsverify --tlscert=/home//.docker/server-cert.pem

--tlscacert=/home//.docker/ca.pem --tlskey=/

home/<user>/.docker/server-key.pem --userns-remap=”default”

-H tcp://0.0.0.0:2376



* After saving the configuration, you will need to restart the Docker service. This can be done by running the following commands:

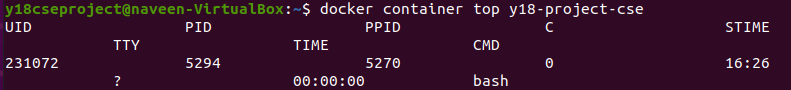
sudo systemctl daemon-reload

sudo systemctl restart docker

* We can now confirm that containers will run under the default dockremap UID. Run a container and give it the name test. Then inspect the container process with this command:

docker container top test

* The output from this command will resemble the following:



**RUNNING DOCKER BENCH FOR SECURITY AFTER SECURING THE DOCKER DAEMON**

Now that we have implemented TLS encryption and user namespaces for containers, we can re-run our security audit with Docker Bench for Security:

sudo ./docker-bench-security.sh

As highlighted in the image below, we should now have an improved security score which highlights the changes we have been making.

Text

Description automatically generated

We have been able to successfully secure the Docker daemon and can begin exploring the various ways of securing Docker containers

**CONTAINER SECURITY BEST PRACTICES:**

Now that we have a secure Docker host and daemon, we can shift our attention to running containers securely. The process of running containers securely is quite robust and will depend on your own use cases. As a result, the topic covers a list of security best practices that you can use when running containers based on your own security requirements.

**USING AN UNPRIVILEGED USER**

Running containers with an unprivileged user will prevent privilege escalation attacks. This can be done by following the outline below:

* + Always reconfigure and build your own Docker images so you can customize the various security parameters to your specification.
  + To run a Docker container as an unprivileged user, you will need to update the Dockerfile before building the image. This can be done by adding a command like the following example to the Dockerfile:

RUN groupadd -r <user> && useradd -r -g <group> <user>

A picture containing text

Description automatically generated

* + This will add the user to the Docker image, and you can now run the container with the unprivileged user instead of running it with the default root user. You can specify the user for a container with the -u option for the docker run command:

docker run -u <user> <IMAGE-ID>

****

**DISABLING THE ROOT USER**

As an added security measure, we can disable the root user of a container by modifying the Dockerfile.

Specifically, we can change the default shell from /bin/bash to /usr/sbin/nologin. This can be done by adding the following command to the Dockerfile:

RUN chsh -s /usr/sbin/nologin root

**DOCKERFILE:**

Graphical user interface, text, application

Description automatically generated

This will prevent any user on the container from accessing the root account regardless of whether they have the root password. This configuration is only applicable if you want to disable the root account completely.

Text

Description automatically generated

**PREVENTING PRIVILEGE ESCALATION ATTACKS**

It is recommended to run your containers with specific permissions and ensure that users cannot escalate their privileges. To do this, use the following flag when running containers:

docker run --security-opt=no-new-privileges <IMAGE-ID>



The no-new-privileges option will stop container processes from gaining any additional privileges. This will prevent commands like su and sudo from working in your container, and it can prevent attacks that exploit SETUID binaries.

**Text

Description automatically generated**

**LIMITING CONTAINER CAPABILITIES**

When you run a container, you can specify a set of kernel capabilities that are available to the container.

Example:

a container can be given the capability of binding to a low-number port on the host (e.g. a web server container that binds to ports 80 and 443).

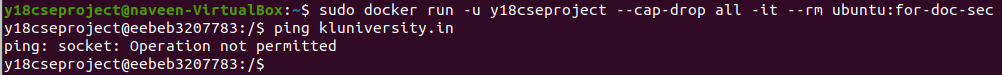
You also can run a container with the --privileged flag, which gives it all the kernel capability options. However, this is never recommended as giving full privileges to a container will usurp any other user permission and security restrictions you have set and open new vulnerabilities.

The recommended method for assigning privileges to a container is to first remove all the capabilities (also referred to as dropping the capabilities) and then only add the ones required for your container to function.

If your container does not need kernel capabilities to run, then they should all be dropped.

* + We can remove all kernel capabilities when running a container with the following options:

docker run --cap-drop all <IMAGE-ID>



* + You can also add the specific kernel capabilities required by your containers by running the following command:

docker run --cap-drop all --cap-add <CAPABILITY> <IMAGE-ID>

In the below image, the capability NET\_RAW is associated with network ( PING), for which the ping command in the above image doesn’t work as the capability NET\_RAW is dropped.

A picture containing graphical user interface

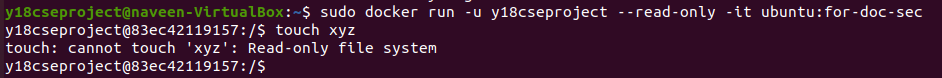
Description automatically generated

**FILE-SYSTEM PERMISSIONS AND ACCESS**

You also have the ability to specify filesystem permissions and access, allowing you to set up containers with a read only file system or with a temporary file system. This option is useful if you would like to control whether your containers can store data or make changes to the filesystem.

* + We can run containers with a read-only file system by running the following command:

docker run --read-only <IMAGE-ID>



* + If your container has a service or application that requires the storage of data, you can specify a temporary file system by running the following command:

docker run --read-only --tmpfs /tmp

**DISABLING INTER CONTAINER COMMUNICATION**

Docker creates a default bridge network, and containers are created on this network by default. All containers on this default network can communicate with each other. However, we can also choose to isolate Docker containers from communicating with one another. For example, this can be helpful if you want to isolate a particular Docker container away from another connected group of containers you’re running.

* + In order to disable inter-container communication, we will need to create a new Docker network. This can be done by running the following command with the “icc” option set to false.

sudo docker network create --driver bridge -o "com.docker.network.bridge.enable\_icc"="false" <NETWORK-NAME>



* + We can now deploy the containers with this network using the below command:

sudo docker run -it --rm --network isolated-network ubuntu:for-doc-sec



**Diagram

Description automatically generated**