Bachelor thesis
Computing Science

Radboud University

Creating a Methodology for Penetration Testing of Docker Containers

Author:
Joren Vrancken
s4593847

Supervisor:
Associate professor, Erik Poll
erikpoll@cs.ru.nl

Internship supervisor:
Dave Wurtz
dave.wurtz@secura.com

Second internship supervisor:
Geert Smelt
geert.smelt@secura.com

October 21, 2019

TODOs are shown like this.

Abstract

Containerization software has become extremely popular to streamline software deployments in the last few years. That has made it a very import attack surface. This paper looks at how one should go about testing the security of the Docker containers. It then provides a methodology for Secura to test the security of Dockers at their clients.

Contents

Chapter 1

Introduction

Secura, a company specializing in digital security, performs security assessments for clients. In these assessments, Secura evaluates vulnerable parts of the private and public network of their clients. They would like to improve those assessments by also looking into containerization software their clients may be running.

Containerization software allows developers to package software into easily reproducible packages. It removes the tedious process of installing the right dependencies to run software, because the dependencies and necessary files are neatly isolated in the container. This also allows multiple versions of the same software to run simultaneous on a server, because every instance runs in its own container.

This thesis will focus on Docker, because it is the de facto industry standard for containerization software. It will focus on Linux, because Docker is developed for Linux (although a Windows version does exist).

This bachelor thesis will first describe necessary background information about containerization, Docker and penetration testing. I will then go into more detail about specific vulnerabilities and misconfigurations that are of interest during a security assessment. Finally, I will describe how a penetration tester can detect and use those vulnerabilities and misconfigurations during security assessments.

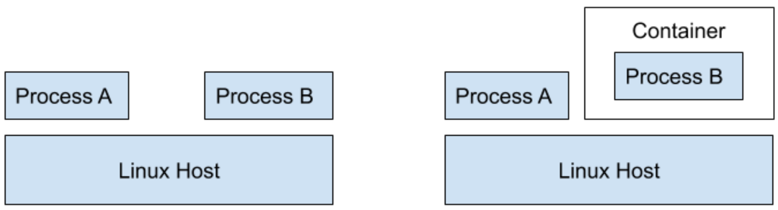
Chapter 2

Background

In this chapter we will talk about necessary background information and preliminaries. First we will look at what containerization software is and how it compares to virtualization. We will also look at important Docker concepts, how to use it and how it works internally. We quickly introduce the CIS Bechmarks. Finally, we look at penetration testing in general and at Secura.

2.1 Containerization Software

Containerization software is used to isolate processes running on a host from one another. A process in a container sees a different part of the host system then processes outside of the container. A process inside a container sees a different file system, network interfaces and users than processes outside of the container. Processes inside the container can only see other processes inside the container.



(a) Two processes (b) One process in a container

If we look at the above example, we see two scenarios. The first is the default way to run processes. The operating system starts processes that can communicate with one another. Their view on the file system is the same. In the second scenario one of the processes runs inside a container. These processes cannot communicate with one another. If Process A looks at the files in `/tmp`, it accesses a different part of the file system than if Process B looks at the files in `/tmp`. Process B can not even see that Process A exists.

Process A and Process B see such a different part of the host system that to Process B it looks like it is running on a whole separate system.

2.1.1 Why use containers?

Containers can be made into easily deployable packages (called images). These images only contain the necessary files for specific software to be run. Other files, libraries and binaries are shared between the host operating system (the system running the container). This allows developers to create lightweight software packages containing only the necessary dependencies.

Containers also make it possible to run multiple versions of the same software on one host. Each container can contain a specific version and all the containers run on the same host. Because the containers are isolated from each other, their incompatible dependencies are not a problem.

For example, someone who wants to run an instance of Wordpress[1] does not need to install all the Wordpress dependencies. They only need to download the container that the Wordpress developers created.

Similarly, if they want to move the Wordpress instance from one host to the other, they just have to copy over their database and run the image on the new host. Even if the new host is a completely different operating system.

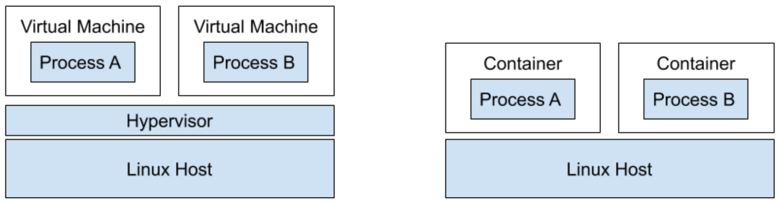
If they want to test a newer version of Wordpress on the same host, they only have to run the different container on the same host. The incompatible dependencies of the two Wordpress instances are not a problem, because they see another part of the file system and do not even see each other's process.

This ease of use makes containerization very popular in software development, maintenance and deployment.

[1] A very popular content management system to build websites with.

Virtualization

Virtualization is an older similar technique to isolate software. In virtualization, a whole system is simulated in top of the host (called the hypervisor). This new virtual machine is called a guest. The guest and the host do not share any system resources. This has some advantages. For example, it allows running a completely different system as guest (e.g. Windows guest run on a Linux host).



(a) Virtual machines (b) Containers

Because containerization software shares many resources with the host, it is a lot faster and more flexible than virtualization. Where virtualization needs to start a whole new operating system, containerization only needs to start a single process.

Containers for security

Isolation reduces risk, because it separates processes. If one process is compromised it cannot reach another process. If a process in a container is compromised, it cannot reach sensitive files of the host. This clearly add security value.

It should be noted, however, that containerization is a lot more risky than virtualization, because containers run using the same kernel and resources as the host. For example, this means that a kernel exploit run inside a container is just as dangerous as the same exploit run directly on the host, because the target (the kernel) is the same.

2.2 Docker

The concept of containerization has been around a long time, but it only gained traction as serious way to package, distribute and run software in the last few years. This is mostly because of Docker.

Docker was released in 2013 and it does not only offer a containerization platform, but also a way to distribute the containers. This allows developers and companies to create packages that have no dependencies (besides Docker itself, of course). This allows for a lot faster development and deployment processes, because dependencies and installation of software are no longer a concern.

Docker also makes it possible to run multiple versions of the same software on the same host, without creating a dependency nightmare. For example, if someone wants to run a Wordpress 4 website and Wordpress 5 website, they only need to create two Wordpress containers. Because the containers are isolated from one another, their conflicting dependencies are not a problem.

2.2.1 Docker Concepts

Docker is exists of a few concepts: Docker daemon, Docker images, Docker containers and `Dockerfile`s.

Docker daemon

The daemon is a service that runs on the host. It manages all things related to Docker on that machine. For example if the user wants to build an image or a container needs to restart the docker daemon. It is good to note that, because everything related to Docker is handled by the daemon and Docker has access to all resources of the host, having access to Docker should be viewed as equivalent to having `root` access to the host[2].

Docker images

A Docker image is packaged software. It is a distributable set of layers. The first layer describes the base of the image. This is either an existing image or nothing (referred to as `scratch`). Each layer on top of that is a change to the layer before. For example, if you add a file or run an command it adds a new layer.

Docker containers

A container is an instance of a Docker Image. If you run software packaged as a Docker image, you create a container based on that image. If you want to run two instances of the same Docker image, you can create two containers.

`Dockerfiles`

A `Dockerfile` describes what a Docker image is made of. It describes the steps to build the image. Lets look at a very simple example:

[2] Docker daemon attack surface

```
FROM alpine:latest
LABEL maintainer="Joren Vrancken"
CMD ["echo", "Hello World"]
```

Listing 2.1: Very Basic `Dockerfile`

These three instructions tell the Docker engine how to create a new Docker image. The full instructionset can be found in the `Dockerfile` reference[3]

1. The `FROM` instruction tells the Docker engine what to base the new Docker image on. Instead of creating an image from scratch (a blank image), we use an already existing image as our basis.
2. The `LABEL` instruction sets a key value pair for the image. There can be multiple LABEL instructions. These key value pairs get packaged and distributed with the image.
3. The `CMD` instruction sets the default command that should be run and which arguments should be passed to it.

We can use this to create a new image and container from that image.

```
$ docker build -t thesis-hello-world .
$ docker run --rm --name=thesis-hello-world-container thesis-
   hello-world
```

Listing 2.2: Creating a Docker container from a `Dockerfile`

We first create a Docker image (called `thesis-hello-world`) using the `docker build` command and then create and start a new container (called `thesis-hello-world-container`) from that image.

Data Persistence

Without additional configuration, a Docker container does not have persistence storage. Its storage is maintained when the container is stopped, but not when the container is removed. It is possible to mount a directory on the host in a Docker container. This allows the container to access files on the host and save them to that mounted directory.

```
(host)$ echo test > /tmp/test
(host)$ docker run -it --rm --volume="/tmp:/tmp" ubuntu:latest
    bash
(cont)$ cat /tmp/test
test
```

Listing 2.3: Bind mount example

[3] `https://docs.docker.com/engine/reference/builder/`

In this example the host `/tmp` directory is mounted into the container as `/tmp`. We can see that a file that is created on the host is readable by the container.

Networking

When a Docker container is created Docker creates a network sandbox for that container and (by default) connects it to an internal bridge network. This gives the container its own networking resources such as a IPv4 address[4], routes and DNS entries. All outgoing traffic is routed through a bridge interface (by default).

Incoming traffic is possible by routing traffic for specific ports from the host to the container. Specifying which ports on the host are routed to which ports on the container is done when a container is created. If we, for example, want to expose port `80` to the Docker image created from the first `Dockerfile` we can execute the following commands.

```
$ docker build -t thesis-hello-world .
$ docker run --rm --publish 8000:80 --name=thesis-hello-world-
   container thesis-hello-world
```

Listing 2.4: Creating a Docker container with exposed port

The first command creates a Docker image using the `Dockerfile` and we then create (and start) a container from that image. We "publish" port `8000` on the host to port `80` of the container. This means that, while the container is running, all traffic from port `8000` on the host is routed to port `80` of the container.

Docker internals

A Docker container actually is a combination of multiple features within the Linux kernel. Mainly `namespaces`, `cgroups` and `OverlayFS`.

`namespaces` are a way to isolate resources from processes. For example, if we add a process to a process `namespace`, it can only see the processes in that `namespace`. This allows processes to be completely isolated from each other. Linux supports the following `namespaces` types[5]:

- `Cgroup`: To isolate processes from `cgroup` hierarchies.
- `IPC`: Isolates the inter-process communication. This, for example, isolates shared memory regions.

[4] IPv6 support is not enabled by default.
[5] See the `man page` of `namespaces`

- `Network`: Isolates the network stack (e.g. IP addresses, interfaces, routes and ports).
- `Mount`: Isolates mount points. When creating a new Mount `namespace` , existing mount points are copied from the current `namespace`. New mount points are not propagated.
- `PID`: Isolates processes from seeing process ids in other `namespaces`. Processes in different `namespaces` can have the same `PID`.
- `User`: Isolates the users and groups.
- `UTS`: Isolates the host and domain names.

When the Docker daemon creates a new container, it creates a new `namespace` of each type for the process that runs in the container. That way the container cannot view any of the processes, network interfaces and mount points of the host. This way it seems that the container is actually an other operating system entirely.

A `mount namespace` is very similar to a `chroot`. A big difference is that a `chroot` has a parent directory. The `mount namespace` can also be more easily combined with other `namespaces` to create more isolation.

Control groups (or `cgroups` for short) are a way to limit resources (e.g. CPU and RAM usage) to (groups of) processes and to monitor the usage of those processes.

`OverlayFS` is a (union mount) file system that allows combining multiple directories and show them as if they are one. This is used to show the multiple layers in an Docker image as a single root directory.

2.2.2 docker-compose

`docker-compose` is a wrapper around Docker that can be used to specify Docker container runtime configurations in files (called `docker-compose.yaml`). These files remove the need to execute Docker commands with the correct arguments in the correct order. You have to specify the necessary arguments only once in the `docker-compose.yaml` file.

This is an advanced example of an `docker-compose.yaml` file similar to configuration that I have used in a production environment. A lot of the time creating Docker containers in production environments, they need to have a lot of extra runtime configuration (e.g. environment variables, ports and dependencies on other containers). Specifying everything in a single file simplifies the runtime configuration process.

```yaml
---
version: "3"

services:
  postgres:
    image: "postgres:10.5"
    restart: "always"
    environment:
      PGDATA: "/var/lib/postgresql/data/pgdata"
    volumes:
      - "/dir/data/:/var/lib/postgresql/data/"

  nextcloud:
    image: "nextcloud:17-fpm"
    restart: "always"
    ports:
      - "127.0.0.1:9000:9000"
    depends_on:
      - "postgres"
    environment:
      POSTGRES_DB: "database"
      POSTGRES_USER: "user"
      POSTGRES_PASSWORD: "password"
      POSTGRES_HOST: "postgres"
    volumes:
      - "/dir/www/:/var/www/html/"
```

Listing 2.5: Example `docker-compose.yaml`

This, however, also shows a security risk. A lot of the information that is passed to the containers is sensitive (e.g. the database password). That information is saved to disk. If the permissions of that file are not set correctly, an attacker could access the sensitive information.

2.2.3 Registries

Docker images are distributable through so called registries. A registry is a server (that anybody can host), that stores Docker images. When a client does not have a Docker image that it needs, it can contact a registry to download that image.

The most popular (and public) registry is Docker Hub, which is run by the same company that develops Docker. Anybody can create a Docker Hub account and start creating images that anybody can download. Docker Hub also provides default images for popular software.

2.3 CIS Benchmarks

`https://docs.docker.com/compliance/cis/docker_ce/`

The Center for Internet Security (or CIS for short) is a non-profit organization that provides best practice solutions for digital security. For example, they provide security hardened virtual machine images that are configured for optimal security.

The CIS Benchmarks are guidelines and best practices on security on many different types of software. These guidelines are freely available for anyone and can be found on their site[6].

They also provide guidelines on Docker[3]. The latest version (1.2.0) contains 115 guidelines. These are sorted by topic (e.g. Docker daemon and configuration files). In the appendix you will find an example guideline from the latest Docker CIS Benchmark.

2.4 Penetration Testing

2.4.1 Methodology Secura

[6] `https://www.cisecurity.org/cis-benchmarks/`

Chapter 3

Known Vulnerabilities & Misconfigurations in Docker

introduction

3.1 Attack Surface & Models

attacks host → container → host

attacks container → host

attacks container → container

Does docker increase/decrease the attack surface of an host

Does docker increase/decrease the impact/likelihood of an exploit?

Linux Capabilities: Secure Your Containers with this One Weird Trick

Research: SELinux

Research: AppArmor

`https://github.com/genuinetools/bane`

Research: Secure Computing Mode Profiles

`https://docs.docker.com/engine/security/security/`

Problem in a deployment pipeline?

`http://training.play-with-docker.com/security-seccomp/`

Namespaces escapes are Docker escapes

Deployment piplines

3.2 Vulnerabilities

Docker CVEs

DOCKER IMAGE VULNERABILITY (CVE-2019-5021)

What to do with non-public CVEs

Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities

Is it possible to escalate privileges and escaping from a Docker container?

VULNERABILITY EXPLOITATION IN DOCKER CONTAINER ENVIRONMENTS

False Boundaries and Arbitrary Code Execution

`https://github.com/gabrtv/shocker`

CVE-2017–5123

CVE-2019-5736-PoC

3.2.1 Alpine Root Password (CVE–2019–5021)

```
$ docker run -it --rm alpine:3.5 cat /etc/shadow
root:::0:::::
```

```
$ docker run -it --rm alpine:3.5 sh
/ # apk add --no-cache linux-pam shadow
...
/ # adduser test
...
/ # su test
Password:
/ $ su root
/ #
```

3.3 Misconfigurations

Docker containers with root privileges

`https://www.katacoda.com/courses/docker-security/`

Permissions on config/service files

Wrong volumes: / or /proc

Non-docker group Docker access?

Research: create user and group in Dockerfile

Map to CIS Benchmark

Does CIS cover everything?

Map this section to CIS Docker Benchmark

Abusing Privileged and Unprivileged Linux Containers

Understanding and Hardening Linux Containers

Securing Docker Containers

10 Docker Image Security Best Practices

3.3.1 The –privileged flag

Understanding Docker container escapes

3.3.2 Root user

How to Run a More Secure Non-Root User Container

Chapter 4

Penetration Testing of Docker

4.1 Manual

4.2 Automated

Source 5-free-tools-to-navigate-through-docker-containers-security

Static analysis tool: `https://github.com/coreos/clair`

Scanner for clair: `https://github.com/arminc/clair-scanner`

Static vulnerability scanner (and clamAV) on software in container: `https://github.com/eliasgranderubio/dagda`

Scanner using the CIS Docker Benchmark: `https://github.com/docker/docker-bench-security`

SaaS container policy scanner: `https://anchore.com`

Research: Twistlock

Research: Sqreen

sysdig: `https://sysdig.com/`

sysdig: `https://sysdig.com/opensource/falco/`

Chapter 5

Future Work

5.1 Kubernetes

Kubernetes Pod Escape Using Log Mounts: `https://blog.aquasec.com/kubernetes-security-pod-escape-log-mounts`

Container Platform Security at Cruise: `https://medium.com/cruise/container-platform-security-7a3057a27663`

An unpatched security issue in the Kubernetes API is vulnerable to a “billion laughs” attack

Basics of Kubernetes Volumes (Part 1)

Basics of Kubernetes Volumes (Part 2)

What is the added value of virtualisation in comparison to containerization?

NIST: Application Container Security Guide

CIS Benchmark Kubernetes

No New Privs

KubiScan

How to Hack a Kubernetes Container, Then Detect and Prevent It

Security Best Practices for Kubernetes Deployment

5.2 Docker on Windows

5.3 Docker Swarm

5.4 Condense Docker CIS Benchmark

The Docker CIS Benchmark contains 115 guidelines with their respective documentation. This makes it a 250+ page document. This is not practical for developers and engineers (the intended audience). It would be much more useful to have a smaller, better sorted list that only contains common mistakes and pitfalls to watch out for.

The CIS Benchmark do indicate the importance of each guideline. With Level 1 indicating that the guideline is a must-have and Level 2 indicating that the guideline is only necessary if extra security is needed. However, only twenty-one guidelines are actually considered Level 2. All the other guidelines are considered Level 1. This still leaves the reader with a lot of guidelines that are considered must-have.

It would be a good idea to split the document into multiple sections. The guidelines can be divided by their importance and usefulness. For example, a three section division can be made.

The first section would describe obvious and basic guidelines that everyone should follow (and probably already does). This is an example of guidelines that would be part of this section:

- 1.1.2: Ensure that the version of Docker is up to date
- 2.4: Ensure insecure registries are not used
- 3.1: Ensure that the docker.service file ownership is set to root:root
- 4.2: Ensure that containers use only trusted base images
- 4.3: Ensure that unnecessary packages are not installed in the container

The second section would contain guidelines that are common mistakes and pitfalls. These guidelines would be the most useful to the average developer. For example:

- 4.4 Ensure images are scanned and rebuilt to include security patches
- 4.7 Ensure update instructions are not use alone in the Dockerfile

- 4.9 Ensure that COPY is used instead of ADD in Dockerfiles
- 4.10 Ensure secrets are not stored in Dockerfiles
- 5.6 Ensure `sshd` is not run within containers

The last section would describe guidelines that are intended for systems that need extra hardening. For example:

- 1.2.4 Ensure auditing is configured for Docker files and directories
- 4.1 Ensure that a user for the container has been created
- 5.4 Ensure that privileged containers are not used
- 5.26 Ensure that container health is checked at runtime
- 5.29 Ensure that Docker's default bridge "`docker0`" is not used

Chapter 6

Related Work

Chapter 7

Conclusions

Docker Security

CIS Benchmarks

Pentesting at Secura

Based on my personal experience

Docker makes applications more secure

Chapter 8

Acknowledgements

Bibliography

[1] Thanh Bui. Analysis of docker security. *CoRR*, abs/1501.02967, 2015.

[2] T. Combe, A. Martin, and R. Di Pietro. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62, Sep. 2016.

[3] Center for Internet Security. CIS docker benchmark. Technical report, Center for Internet Security, 07 2019.

[4] A. Martin, S. Raponi, T. Combe, and R. Di Pietro. Docker ecosystem – vulnerability analysis. *Computer Communications*, 122:30 – 43, 2018.

[5] Rui Shu, Xiaohui Gu, and William Enck. A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 269–280, New York, NY, USA, 2017. ACM.

Either cite or remove Bibliography entries

Appendix A

Example guideline from Docker CIS Benchmark

4.8 Ensure setuid and setgid permissions are removed (Not Scored)

Profile Applicability:

- Level 2 - Docker - Linux

Description:

Removing setuid and setgid permissions in the images can prevent privilege escalation attacks within containers.

Rationale:

setuid and setgid permissions can be used for privilege escalation. Whilst these permissions can on occasion be legitimately needed, you should consider removing them from packages which do not need them. This should be reviewed for each image.

Audit:

You should run the command below against each image to list the executables which have either setuid or setgid permissions:

```
docker run <Image_ID> find / -perm /6000 -type f -exec ls -ld {} \; 2>
/dev/null
```

You should then review the list and ensure that all executables configured with these permissions actually require them.

Remediation:

You should allow setuid and setgid permissions only on executables which require them. You could remove these permissions at build time by adding the following command in your Dockerfile, preferably towards the end of the Dockerfile:

```
RUN find / -perm /6000 -type f -exec chmod a-s {} \; || true
```

Impact:

The above command would break all executables that depend on setuid or setgid permissions including legitimate ones. You should therefore be careful to modify the command to suit your requirements so that it does not reduce the permissions of legitimate programs excessively. Because of this, you should exercise a degree of caution and examine all processes carefully before making this type of modification in order to avoid outages.

134 | Page

Default Value:

Not Applicable

References:

1. http://www.oreilly.com/webops-perf/free/files/docker-security.pdf
2. http://container-solutions.com/content/uploads/2015/06/15.06.15_DockerCheatSheet_A2.pdf
3. http://man7.org/linux/man-pages/man2/setuid.2.html
4. http://man7.org/linux/man-pages/man2/setgid.2.html

CIS Controls:

Version 6

5.1 Minimize And Sparingly Use Administrative Privileges
Minimize administrative privileges and only use administrative accounts when they are required. Implement focused auditing on the use of administrative privileged functions and monitor for anomalous behavior.

Appendix B

Interview Questions

Penetration Testing

- What is the Penetration Testing methodology of Secura?

Docker

- Do you know what Docker is?
- Have you ever encountered Docker during an assessment?
- Do you actively look for Docker in client networks?
- Have you ever reported an issue about Docker for a client?
- Do you think Docker makes applications/systems more secure?