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 3, 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

Not about app vulnerabilities, but specifically about Docker

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.
The de facto industry standard is called Docker. Docker allows developers to package software into images and run those instances as containers.
Docker streamlines and significantly simplifies software development and deployment. That is why many companies use Docker to develop, test and deploy (part of) their IT infrastructure someway or another. This makes it very interesting from a security perspective. A security problem with Docker could have a large impact on organizations.

This research paper describes possible security problems with Docker (vulnerabilities and misconfigurations) and how those can be used during security assessments.

Chapter 2

Background

2.1 Containerization Software

Democratization of Container Technologies

A Brief History of Containers: From the 1970s to 2017

Containerization software is used to isolate software into packages (called containers). These containers 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.

This is great for rapid development of software, because every small change can quickly be packaged into a container.
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.

This has made containerization a very popular concept in developing and running software.

Virtualization

Virtualization is an older technique used 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).

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.

Where virtulization can and is used as a security layer, because it truly isolates the host and guest resources. Containerization should not be used as a security layer, because containers might be able to access sensitive resources that are shared between the host and the container. This makes containerization a lot more dangerous from a security perspective.

2.2 Docker

Docker on Windows

Research: Secure Computing Mode Profiles

chroot, cgroups and namespaces

Docker, A brief history and security considerations for modern environments

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 did not only offer a containerization platform, but also a way to distribute the containers. This allows developers and companies to create packages that are much more easily run.

For example, someone that wants to run an instance of Wordpress[1], does not need to install all the Wordpress dependencies. They only need to download the Docker image 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 Docker image on the new host. Even if the new host is a completely different operating system.

2.2.1 Docker Concepts

Docker is based on four concepts: Docker daemon, Docker images, Docker containers and `Dockerfiles`.

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

Docker daemon

The daemon is a service that runs on the host machine. 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:

```
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.

[2] `https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface`

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

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.

Networking

iptables

https://github.com/docker/libnetwork/blob/master/docs/design.md

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 2.3: Creating a Docker container with exposed port

[4] IPv6 support is not enabled by default.

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

Demystifying Containers Part I Kernel Space

Demystifying Containers Part II Container Runtimes

Demystifying Containers Part III Container Images

2.2.2 docker-compose

Secrets in docker-compose can be easily found, even without docker group permissions

2.2.3 Registries

Official images not always standard images

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 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 Penetration Testing

2.3.1 Methodology Secura

2.4 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[5].

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

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

Chapter 3

Known vulnerabilities in Docker

Map this chapter to CIS Docker Benchmark

Top 5 Docker Vulnerabilities You Should Know

Docker containers with root privileges

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

Hacking Containers and Kubernetes

3.1 Vulnerabilities

3.2 Misconfigurations

Permissions on config/service files

Wrong volumes: / or /proc

Non-docker group Docker access?

Research: create user and group in Dockerfile

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`

Basics of Kubernetes Volumes (Part 1)

Basics of Kubernetes Volumes (Part 2)

5.2 Condense CIS Benchmarks

The CIS Benchmarks for Docker 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 Benchmarks 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

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 CIS benchmarks

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?