AARHUS UNIVERSITET

Microservices and DevOps

Scalable Microservices

Docker Security

Henrik Bærbak Christensen

AARHUS UNIVERSITET

Introduction

- Sources:
  - Newman §9
    - Some aspects has already been covered
  - Snyk.io / Vermeer
    - A few aspects have been covered
  - Docker secrets
    - Find it on Docker website

AARHUS UNIVERSITET

Security in Transit

- Security in transit
  - Use TLS to ensure encrypted (at least outwards) traffic
  - Use Authentication and Authorization
    - Like OAuth 2.0
  - Deputy problem is relevant in a MicroService context:
    - Cmd calls Daemon *calls CaveService*
      - But cmd is only authorized against Daemon?
  - Solution: OAuth 2.0 provides the /introspect feature
    - Daemon hands bearer token to CaveService
    - CaveService can then /introspect validity on the AuthServer
      - And thus establish trust without trusting Daemon

AARHUS UNIVERSITET

Security at Rest

- Encrypt the databases…
  - For Redis, it seems it relies on encrypted file systems !
    - I.e. no direct support by the Redis DB itself
      - As far as I can dig out of the documentation
  - Do not invent encryption algorithms yourself ☺
    - Use existing tools and algorithms
    - Use BitLocker, etc.
  - For passwords, store them using *salted password hashing*
    - Actually the jCrypt (BCrypt implementation) is used even in the Stub subscription service

```
public SubscriptionPair(String password, SubscriptionRecord record) {
  String salt = BCrypt.gensalt( log_rounds: 4); // Preferring faster over
  String hash = BCrypt.hashpw(password, salt);

  this.bCryptHash = hash;
  this.subscriptionRecord = record;
}
```

AARHUS UNIVERSITET

Defense in Depth

- Firewalls
  - Only allow network traffic on proper ports
    - i.e. **no “ports 6379” on your Redis in the swarm**
    - It is already accessible from with the swarm but now you expose it to the outside!
  - Beware of issuing ‘docker run –p 6379:6379…’ on a production machine
    - Docker will open the port using IPTABLES and ‘ufw’ does not see it!

AARHUS UNIVERSITET

Defense in Depth

- Logging
  - Proper logging allows detecting and recovering from attacks
  - Do not store sensitive information in logs
    - As the skycave daemon does at the moment!
- Intrusion Detection Systems
  - IPS can help out
    - I have no personal experience ☹

AARHUS UNIVERSITET

Defense in Depth

- Network Segregation
  - Separate nodes onto separate networks
  - I.e. a ‘frontend-network’ for the API Gateway
  - And a ‘backend-network’ for the internal services + databases
- Swarm is excellent at this
  - You can attach a service on multiple networks
    - API gateway on both frontend and backend
- DigitalOcean et al.
  - Provides ‘private network’
    - Use that network for Cluster communication, not the official IP address

AARHUS UNIVERSITET

Not mentioned?

- Principle of *least privilege* (Source: Wikipedia)

> In information security, computer science, and other fields, the **principle of least privilege** (**PoLP**), also known as the **principle of minimal privilege** or the **principle of least authority**, requires that in a particular abstraction layer of a computing environment, every module (such as a process, a user, or a program, depending on the subject) must be able to access only the information and resources that are necessary for its legitimate purpose.[1]

- In Linux
  - Control your 'umask'
  - Ensure only you can read/write sensitive files
  - Don't be the root user
  - Etc.

```
csdev@m1-dev:~/.ssh$ ll
total 24
drwx------  2 csdev csdev 4096 Sep 26  2020 ./
drwxr-xr-x 42 csdev csdev 4096 Nov 19 10:17 ../
-rw-------  1 csdev csdev 3326 Nov 17  2017 id_rsa
-rw-rw-r--  1 csdev csdev  398 Sep 26  2020 id_rsa.pub
-rw-r--r--  1 csdev csdev 5532 Oct 25 12:53 known_hosts
```

AARHUS UNIVERSITET

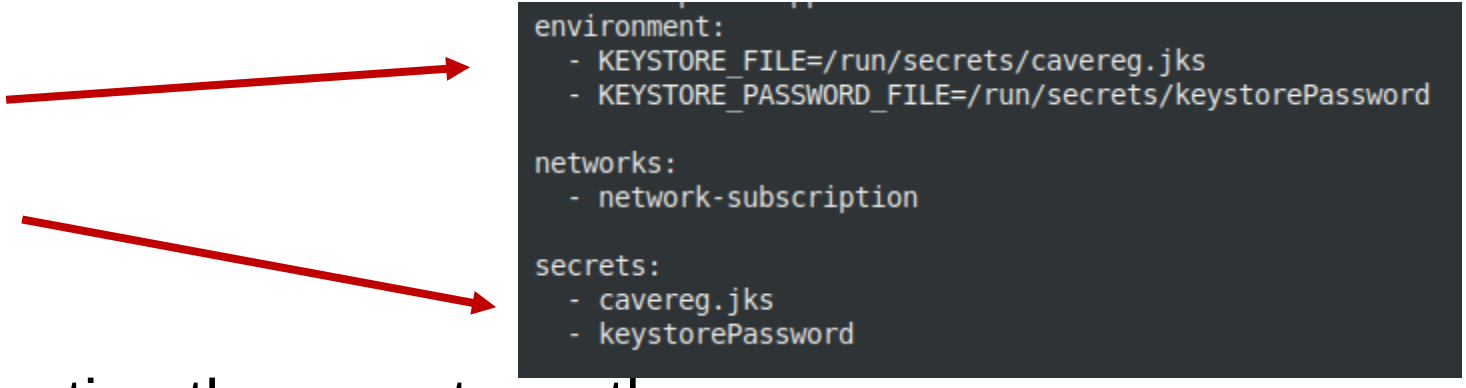
Key Management

- Docker Swarm has excellent secure key management
  - Usernames and passwords
  - TLS certificates and keys
  - Database passwords, etc
- ***Only works in stacks*** – not in standalone containers
- Idea
  - *docker secret create thesecret hostfile*
    - Will
      - Encrypt local ‘hostfile’
      - Send it to all nodes in swarm using TLS
      - Make file available in /run/secrets/thesecret
  - Passwords are a bit weird, as secrets are *files…*

AARHUS UNIVERSITET

Docker Secret

- Snippet from Subscription Service's compose file



```
environment:
  - KEYSTORE_FILE=/run/secrets/cavereg.jks
  - KEYSTORE_PASSWORD_FILE=/run/secrets/keystorePassword

networks:
  - network-subscription

secrets:
  - cavereg.jks
  - keystorePassword
```

- And creating the secrets on the swarm manager

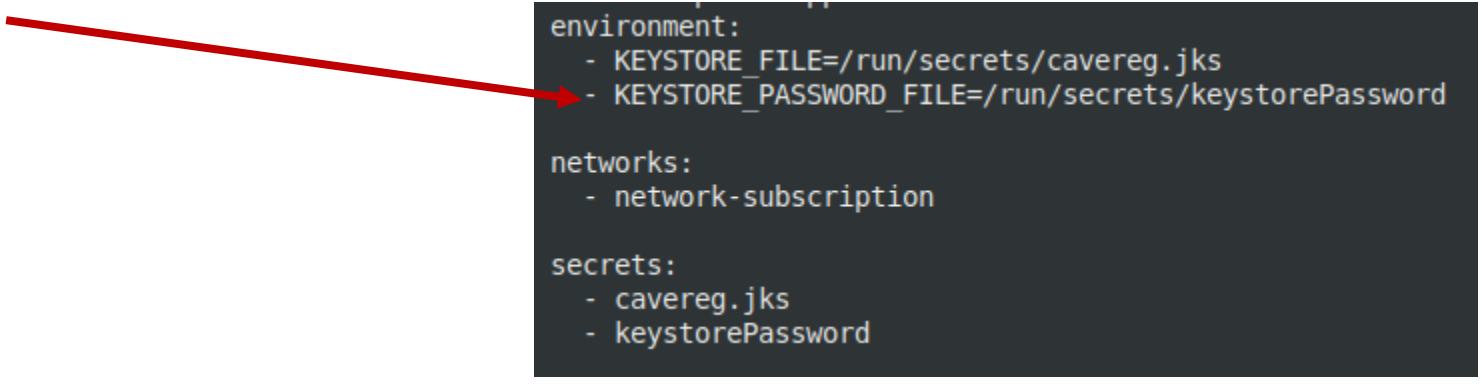
```
echo "changeit" | docker secret create keystorePassword -
docker secret create cavereg.jks ~/secrets/cavereg.jks
```

  - i.e. you have to *fiddle* with the server to set the passwords
    - Fair enough, but well manual process

AARHUS UNIVERSITET

Reading ‘passwords’

- As all secrets are files – what to do with a password?
  - You have to fiddle with a ENTRYPOINT script



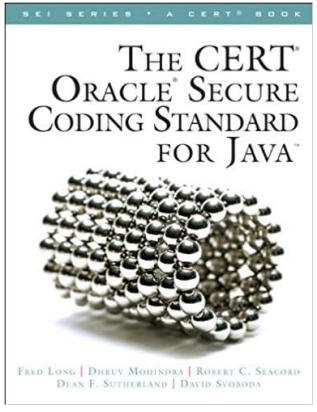
```
environment:
  - KEYSTORE_FILE=/run/secrets/cavereg.jks
  - KEYSTORE_PASSWORD_FILE=/run/secrets/keystorePassword

networks:
  - network-subscription

secrets:
  - cavereg.jks
  - keystorePassword
```

```
if test -f "$KEYSTORE_PASSWORD_FILE"; then
    echo "Picking keystore password from a docker secret file"
    # Read in the password in the docker secret and set the env accordingly
    export KEYSTORE_PASSWORD=$(cat "$KEYSTORE_PASSWORD_FILE")
else
    echo "Keystore password set from -e switch"
fi
```

Vulnerabilities

- All code can contain loopholes that attackers can use
  - Code vulnerabilities
- Writing secure (Java) code…
  - Is a major topic on its own

    And way outside scope of course and my area of expertise…



- Ensure you do not import other's vulnerabilities
  - I.e. *ensuring your container is as secure as possible*

AARHUS UNIVERSITET

Snyk.io

- You can scan your image for vulnerabilities

```
csdev@m1-dev:~/proj/crunch4$ docker scan henrikbaerbak/jdk11-gradle68
```

```
Tested 232 dependencies for known issues, found 82 issues.

Base Image              Vulnerabilities  Severity
ubuntu:bionic-20201119  40               0 critical, 1 high, 13 medium, 26 low

Recommendations for base image upgrade:

Minor upgrades
Base Image    Vulnerabilities  Severity
ubuntu:18.04  24               0 critical, 0 high, 3 medium, 21 low

Major upgrades
Base Image              Vulnerabilities  Severity
ubuntu:impish-20211015  12               0 critical, 0 high, 2 medium, 10 low
```

- ☹

AARHUS UNIVERSITET

Vermeer (2021)

- Vermeer provide a set of practices to use in creating images, which lowers the amount of vulnerabilities.
- I tried them and got ☺ this…

```
csdev@m1-dev:~/proj/cave$ docker scan henrikbaerbak/private:cave-jar

Testing henrikbaerbak/private:cave-jar...

Organization:      henrikbaerbak
Package manager:   apk
Project name:      docker-image|henrikbaerbak/private
Docker image:      henrikbaerbak/private:cave-jar
Platform:          linux/amd64
Base image:        alpine:3.14.3
Licenses:          enabled

✓ Tested 18 dependencies for known issues, no vulnerable paths found.

According to our scan, you are currently using the most secure version of the selected base image
```

AARHUS UNIVERSITET

Advice in short form

- Source: 10 best practices to containerize Java applications with Docker

1. Use deterministic docker base image tags

- Avoid `FROM openjdk`
- Avoid `FROM maven:openjdk`
- Avoid `FROM maven:3-jdk11`

Instead of generic image aliases, use SHA256 hashes or specific image version tags for deterministic builds. For example:

```
FROM maven:3.6.3-jdk-11-slim@sha256:68ce1cd457891f
```

2. Install only what you need in production

You do not need a JDK, the Java code, and a build tool like Maven or Gradle in your production image. Instead, use the product of your Java build.

- Copy the Jar or War.
- Use a Java Runtime Environment (JRE).

3. Find and fix security vulnerabilities in your Java Docker image

Docker base images may include security vulnerabilities in the software toolchain they bundle, including the Java Runtime Environment itself.

Scan and fix security vulnerabilities with the free Snyk Container tool which also provides base image recommendations:

```
npm install -g snyk
snyk auth
snyk container test myimage
--file=Dockerfile
```

4. Use multi-stage builds

Separate your building image from your production image.

- Build your artifacts in the build stage with all possible tools and secrets you need.
- Copy the resulting artifact(s) to the most minimal production image.

5. Don't run Java apps as root

Docker defaults to running the process in the container as the root user, which is a precarious security practice. Use a low privileged user and proper filesystem permissions:

- Create a new user.
- Give the user only necessary permissions.
- Call `USER youruser`.

6. Properly handle events to safely terminate a Java application

Docker creates processes—such as PID 1—and they must inherently handle process signals to function properly. This is why you should avoid any of these variations:

- `CMD "java" "-jar" "application.jar"`
- `CMD "start-app.sh"`

Instead, use a lightweight init system, such as dumb-init, to properly initialize the Java process with signals support:

```
CMD "dumb-init" "java" "-jar"
"application.jar"
```

7. Gracefully tear down Java applications

Avoid an abrupt termination of a running Java application that halts live connections;. either use an application server or create a shutdown hook. Try using a process signal event handler:

```
Runtime.getRuntime().addShutdownHook
(yourShutdownThread);
```

8. Use .dockerignore

Use `.dockerignore` to ensure:

- no debug log files appear in your container.
- no secrets are accidentally leaking.
- a small Docker base image without redundant and unnecessary files.

9. Make sure Java is container-aware

Old JVM versions don't respect Docker memory and CPU settings. Make sure the JVM is container-aware.

- Use Java 10+
- Use Java 8 update 191+

10. Be careful with automatic Docker container generation tools

Tools like Spring Boot 2.3 Docker image creation and JIB are great tools to automatically build Docker images. However, you are not aware of all security concerns. So, be careful when using these!

Instead, consider creating a specific Dockerfile implementing all best practices.

AARHUS UNIVERSITET

Summary

- Lots of attack surfaces ☹
  - Lots of place in which you *must do the right thing* and *adhere to the right protocols and practices*