IT-UNIVERSITETET I KØBENHAVN **DevOps**

Weapons - Final Report

May 23, 2023

Author 1
Author 2
Author 3

Contents

1 System Perspective

Lorem ipsum dolor Lorem ipsum dolor

2 Process Perspective

2.1 Security analysis

2.1.1 Risk Identification

Identify Assets

- Web application
- Dockerhub
- GitHub
- Digital Ocean
- Grafana / Prometheus
- SonarCube
- Teams

Identify Threat Sources

- SQL injection (very relevant)
- Password leakage
- GitHub is public (avoid committing sensitive information)
- Unused ports are closed off / behind capable firewalls
- Admin passwords and their strength on all relevant platforms
- Vulnerability of used platforms / frameworks
- Malicious code / images on Dockerhub

- Individual passwords for GitHub
- Teams channel is full of sensitive things
- Access to personal developer computers

Construct Risk Scenarios

An attacker could attempt a SQL injection with the goal of downloading the database, or parts hereof. This would expose the stored user information – unhashed email addresses and encrypted passwords – potentially harming users.

Suppose a developer commits sensitive information – such as passwords or ssh keys – to the public GitHub repository. An attacker then uses this information for their malicious intents, which would be almost limitless with root access.

A developers personal computer is breached because they have low strength passwords, allowing attackers immediate access to server, GitHub, DockerHub, pushing malicious code, and many other things.

2.1.2 Risk Analysis

Determine Likelihood, Impact & Potential discussions

- SQL Injection
  - As we utilize the prepared statements found in the Java interface SQL, the likelihood is almost entirely bound by the risk of vulnerabilities in this interface. Outside of this, the risk of SQL injections are extremely low, as we only ever query with the above statements.
  - The impact would be moderate. The attackers would be able to retrieve emails and hashed passwords. However, since the passwords are salted and hashed, the password leak would not be detrimental to users. The leaking of emails is a privacy concern.
  - Should this happen, users must be informed that their emails were potentially leaked, and that their passwords should still be save, as we have stored them responsibly (in that we don't store them).
- Password leakage
  - As touched upon above, we salt and hash the passwords, and do not store any actual passwords. The risk here is the same a above.
  - Should the stored password hash be leaked, it would be of negligible impact.
- Public GitHub
  - As this is dependent upon the developers, who are all currently humans, the likelihood is possible bordering on likely.
  - The impact of such a mistake, can in worst case scenarios be severe. A private key being available on a public repository is about as severe as possible.
  - As this scenarios actually happened to us. Early on, a private ssh key was pushed to the repository. As a response to this, the key should be changed immediately, and talks with developers, concerning how secrets are handled, should take place as soon as possible. Depending on the severity of what was leaked, a full code review should take place, to identify possible newly introduced vulnerabilities.
- Port & Firewalls
  - The likelihood a port is mistakenly open or a hole in the firewall exists is possible.

  - The impact is dependent on what is exposed said port, or said hole in the firewall. The range of this, is the entire spectrum of impact severity.
  - Once a hole or unintended port is found, both should be closed, and depending on what was exposed, a review of malicious changes should happen.

- Shared admin password being leaked/cracked
  - As the admin passwords have been shared on the teams channel, the likelihood is dependent on many factors, however, so guesstimate it here, we put it as possible. It could potentially be "likely" due to the many different attack surfaces that could expose it; teams accounts, personal computers, and the password itself.
  - Severe++.
  - There should not be a single root password. Instead, developers that need to have admin access, should have their own user that has 2 factor authentication as well as a significantly strong password, with the necessary permissions. Should a developer's user be breached, then the individual users make it simple to identify what changes the breached user made, if any, as well as easily and quickly restrict its access while maintaining the other developers'.

- Vulnerability in used platforms / frameworks
  - The likelihood here is arguably likely, if not very likely, simply due to the amount of dependencies a project like this relies on.
  - The impact depends on what platform and or framework has a vulnerability, and how the project uses it.
  - In reality, the likelihood can be kept nearer unlikely by staying up to date with all patches and updates to said dependencies.

- Image tampering on Dockerhub
  - The likelihood of this depends on the likelihood of an attacker getting our Dockerhub credentials, and/or an adversary is capable of acting as a man-in-the-middle and intercepting our messages to Dockerhub and replacing messages with malicious images. For the first scenario, see the admin password section, and for the latter, very unlikely as the communication between us and Dockerhub is encrypted with TLS.
  - Severe, as it allows arbitrary code execution.

- Developer passwords and their strengths
  - As we currently have no policy for neither password strength nor two factor authentication, the likelihood must be at least likely.
  - Severe, as it gives essentially arbitrary code execution.
  - All on the project should use two factor authentication as well as a strong password.

- Teams channel
  - As the teams channel is accessible with each developers Microsoft account, and as mentioned above we have no policy for securing such credentials, it's also at the very least likely.
  - Since some secrets have been shared through this channel, the impact could be severe.
  - Don't share secrets in a teams channel. It's a teams channel...

- Personal developer computers and the risks associated.

- Unlikely. The likelihood is directly related to how careful the developers are with their computer. However, it also requires physical access, as well as breaking into the locked machine. This does not necessarily make it more difficult or unlikely, but it is a significant hurdle, unless strongly motivated as it also carries a higher risk for the attacker.
- Should all of the above succeed for the attacker, the impact cannot be less than severe.

2.1.3 Pen-testing

Using the OWASP ZAP program, we were able to identify a short list of possible security issues with our own application. Of this list, we decided to begin by fixing the Content Security Policy header. In our application, we had not at this point made any CSP, and, therefore, a vulnerability to cross site scripting (XSS) and data injection attacks, to list a few.

As our application uses the Spring framework, it felt natural to use the Security tools Spring provides. In the class ContentSecurityPolicyConfiguration.java, we dictate our CSP. The actual policy used, is a policy found online that should cover the most basic issues.

The goal with setting this policy, is to limit the things that a users' browser will execute. For example, we only allow image sources to come from *.gravatar.com, which is the website we get user avatars from. Any other image won't be load by the browser. The interesting thing with CSP's is, we are not limiting what the application will do, but instead limiting what the users' browser will do. This is why it's a useful tool against XSS, as we can limit what scrips can and cannot be executed.

3 Lessons Learned

Lorem ipsum dolor Lorem ipsum dolor