Institute of Software Technology
Reliable Software Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

# Securing DevOps — Detection of vulnerabilities in CD pipelines

Christina Paule

**Course of Study:** Softwaretechnik

**Examiner:** Dr.-Ing. André van Hoorn
**Supervisors:** Thomas Düllmann, M.Sc.,
University of Stuttgart
Andreas Falk, Managing Consultant,
NovaTec Consulting GmbH
Andreas Reinhardt, Senior Consultant,
NovaTec Consulting GmbH

**Commenced:** October 19, 2017
**Completed:** April 19, 2018

# Abstract

Nowadays more and more companies implement the DevOps approach. DevOps was developed to enable more efficient collaboration between development (dev) and operation (ops) teams. An important reason why companies use the DevOps approach is that they aspire to continuously deliver applications using agile methods. The continuous delivery (CD) process can be achieved with the aid of the DevOps approach, of which the CD pipeline is an elementary component.

Because of the fact, that a new General Data Protection Regulation (GDPR) will enter into force in the European Union in May 2018, many companies are looking at how they can increase the security level of their applications. The regulation requires that companies which process personal data have to secure their applications. An attacker can gain access to personal data if there are vulnerabilities in applications. This problem can be applied to CD pipelines. If CD pipelines have vulnerabilities then an exploitation of vulnerabilities can lead to a damage of the CD pipeline and the delivery process. One example is that the network can be scanned by running injected malicious unit tests. This can have a negative effect on the image of the company which operates and uses CD pipelines. Therefore, the question arises which vulnerabilities are present in CD pipelines and how they can be detected.

The theoretical findings of this research are extended by a practical case study. The aim of the case study is to find out how secure the industry CD pipelines are. This aim is achieved by identifying the vulnerabilities in these CD pipelines. The knowledge of developers is used to narrow down this topic. This knowledge is obtained by a survey on which 19 employees of a specific company have participated. The results show, that these developers perform tasks with the CD pipeline, but they rarely deal with security aspects. From the view of developers, the CIA security attributes (confidentiality, integrity, and availability) are the most important ones to consider. To detect vulnerabilities in CD pipelines, the STRIDE threat method was executed on a generalized CD pipeline. The results show that unencrypted connections between CD pipeline components and unrestricted access are possible vulnerabilities for this CD pipeline. Tools are required for continuous automatic detection of the theoretical explored vulnerabilities. The results of this thesis show that there exists at least one tool for the detection or mitigation of vulnerabilities in each CD pipeline stage and component. In the aforementioned case study, the investigation of two CD pipelines which are used in a selected company is performed. The results of this investigation show that both CD pipelines include vulnerabilities which have potentially high risks. The company will try to mitigate all detected vulnerabilities. The mitigation of all vulnerabilities is partly difficult because the two project teams are dependent on the available budget, time, and the provided infrastructure of the customer.

This thesis can be used to show companies and organizations with similar CD

pipelines how they can reduce the overall risk severity of their CD pipelines. In this thesis, someone can find out which possible vulnerabilities exist in CD pipelines and how they can be detected and mitigated through tools.

# Kurzfassung

Heutzutage setzen immer mehr Unternehmen den DevOps-Ansatz ein. DevOps wurde entwickelt, um eine effizientere Zusammenarbeit zwischen Entwicklung (dev) und Betrieb (ops) zu ermöglichen. Ein wichtiger Grund, warum Unternehmen den DevOps-Ansatz nutzen, ist die Forderung, Anwendungen kontinuierlich mit agilen Methoden bereitzustellen. Der kontinuierliche Auslieferungsprozess kann durch DevOps-Methoden erreicht werden, von denen die CD Pipeline ein elementarer Bestandteil ist.

Im Mai 2018 wird in der Europäischen Union eine neue allgemeine Datenschutzverordnung (GDPR) in Kraft treten, wodurch sich viele Unternehmen damit beschäftigen, wie sie das Sicherheitsniveau ihrer Anwendungen erhöhen können. Die Verordnung verlangt, dass jedes Unternehmen, das personenbezogene Daten verarbeitet, ihre Anwendungen sichern muss. Ein Angreifer kann Zugriff auf persönliche Daten erhalten, wenn es Schwachstellen in der Anwendung gibt. Dieses Problem kann auf CD Pipelines übertragen werden. Sofern CD Pipelines Schwachstellen aufweisen und ein Angreifer die Möglichkeit besitzt diese auszunutzen, kann dies dazu führen, dass der Angreifer die CD Pipeline und den Auslieferungsprozess beschädigt. Ein Beispiel ist, dass das Netzwerk gescannt werden kann, indem injizierte bösartige Unit-Tests ausgeführt werden. Eine mögliche Auswirkung hiervon ist, dass das Image des Unternehmens, welches die CD Pipelines betreibt und nutzt, beschädigt wird. Daher stellt sich die Frage, welche Schwachstellen in CD Pipelines vorhanden sind und wie diese erkannt werden können.

Die theoretischen Erkenntnisse dieser Forschung werden durch eine Fallstudie ergänzt. Ziel der Fallstudie ist es herauszufinden, wie sicher die Industrie CD Pipelines sind. Die Erreichung dieses Zieles erfolgt durch die Identifizierung der Schwachstellen in diesen CD Pipelines. Das Wissen der Entwickler wird dazu verwendet, um das Thema einzugrenzen. Eine Umfrage, an der 19 Mitarbeiter einer auserwählten Firma teilgenommen haben, wurde durchgeführt, um diese Kenntnisse zu erlangen. Das Ergebnis zeigt, dass diese Entwickler Aufgaben mit der CD Pipeline erledigen, aber sich nur selten mit Sicherheitsaspekten beschäftigen. Aus Sicht der Entwickler sind die CIA-Sicherheitsattribute (Vertraulichkeit, Integrität, Verfügbarkeit) die Wichtigsten, die betrachtet werden sollen. Um Schwachstellen in CD Pipelines zu erkennen, wurde auf dieser Grundlage die STRIDE-Bedrohungsmethode auf einer allgemeinen CD Pipeline ausgeführt. Die Ergebnisse zeigen, dass unverschlüsselte Verbindungen zwischen CD Pipeline Komponenten und uneingeschränkte Zugriffe potenzielle Schwachstellen der CD Pipeline sind. Zur kontinuierlichen automatischen Erkennung der erkannten Schwachstellen werden Werkzeuge benötigt. Die Forschung nach diesen Werkzeugen zeigt, dass für jede CD Pipeline-Stufe und Komponente mindestens ein Werkzeug zur Schwachstellenerkennung oder -minimierung existiert. In der obengenannten Fallstudie wird die Untersuchung von zwei CD Pipelines (A und B), die in einer ausgewählten Firma

eingesetzt werden, durchgeführt. Die Ergebnisse dieser Untersuchung zeigen, dass beide CD Pipelines Schwachstellen mit einer potenziell hohen Gesamtrisikostärke aufweisen. Das Unternehmen wird versuchen, alle erkannten Schwachstellen zu minimieren. Die Behebung aller Schwachstellen ist teilweise schwierig, da die beiden Projekt-Teams von dem verfügbaren Budget, der Zeit und der bereitgestellten Infrastruktur des Kunden abhängig sind.

Mit dieser Arbeit kann einem Unternehmen und einer Organisation, die ähnliche CD Pipelines einsetzen, gezeigt werden, wie sie die Gesamtrisikostärke ihrer CD Pipelines reduzieren können. Darüber hinaus kann herausgefunden werden, welche potenziellen Schwachstellen in CD Pipelines vorhanden sind und wie diese durch Werkzeuge erkannt und abgemildert werden können.

# Contents

1. Introduction 1
1.1. Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2. Research goal and research questions . . . . . . . . . . . . . . . . . . . 2
1.3. Research methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.4. Thesis structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2. Foundations 7
2.1. Continuous delivery (CD) and continuous deployment . . . . . . . . . . 7
2.2. CD pipeline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.3. Development and operations (DevOps) . . . . . . . . . . . . . . . . . . 10
2.4. Security . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.4.1. Information security . . . . . . . . . . . . . . . . . . . . . . . . 11
2.4.2. DevSecOps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
2.5. Vulnerabilities . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
2.5.1. Vulnerabilities and the relation to threat, asset and risk . . . . . . 13
2.5.2. Threats, attributes, and means for attaining dependability and security . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
2.5.3. Software vulnerabilities . . . . . . . . . . . . . . . . . . . . . . . 15
2.5.4. Pipeline vulnerabilities . . . . . . . . . . . . . . . . . . . . . . . . 16
2.5.5. Open Web Application Security Project (OWASP) and software weakness lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
2.5.6. Vulnerability detection approaches . . . . . . . . . . . . . . . . . 17
2.6. Attacks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.6.1. Attacks based on architecture and design . . . . . . . . . . . . . . 22
2.6.2. Attacks during operation . . . . . . . . . . . . . . . . . . . . . . . 23
2.6.3. Attacks based on authorization . . . . . . . . . . . . . . . . . . . 24
2.6.4. Attacks on CD pipeline stages and components . . . . . . . . . . 24
2.7. DevSecOps and static analysis tools . . . . . . . . . . . . . . . . . . . . 26

3. Related work 33
3.1. Securing web applications . . . 33
3.2. Securing pipelines . . . 34
3.3. Threat modeling . . . 36
3.4. DevSecOps . . . 37

4. Survey on the knowledge of developers 41
4.1. Qualitative research method . . . 41
4.2. Survey design and questions . . . 42
4.3. Survey execution . . . 43
4.4. Survey results . . . 44
4.4.1. The results of the questions from the second part of the survey . 44
4.4.2. The results of the questions from the first part of the survey . . . 47

5. Vulnerabilities in CD pipelines 51
5.1. Prerequisites for vulnerability detection . . . 51
5.2. Threat and vulnerability detection with STRIDE . . . 53
5.3. Detection of vulnerabilities in Jenkins . . . 66
5.3.1. Change Jenkins configuration file without notification and Jenkins authorization . . . 66
5.3.2. Change Jenkins security properties without notification . . . 67
5.3.3. Remove relevant files with Jenkins . . . 68
5.4. Summary of vulnerabilities in CD pipelines . . . 68

6. Tools for securing CD pipelines 71
6.1. Research method and literature foundations . . . 71
6.2. Detection of vulnerabilities in CD pipeline files . . . 72
6.3. Detection of vulnerabilities in CD pipeline components . . . 74
6.4. Detection of sensitive data in commit messages . . . 74
6.5. Detection of vulnerabilities in pipeline configurations . . . 75
6.6. Detection of vulnerabilities in application dependencies . . . 77
6.7. Detection of vulnerabilities in Docker containers/images . . . 79
6.8. Detection of vulnerabilities in artifacts . . . 82
6.9. Detection of vulnerabilities in infrastructure . . . 83
6.10. Detection of vulnerabilities with monitoring and logging tools . . . 84
6.11. Summary overview of the selected tools and detected vulnerabilities . . 86

7. Case Study 91
7.1. Case study design and planning . . . 91
7.1.1. Rationale for the study . . . 92
7.1.2. Objective of the study . . . 92

7.1.3. Cases and units of analysis . . . . . . . . . . . . . . . . . . . . . . . . 92
7.1.4. Research questions . . . . . . . . . . . . . . . . . . . . . . . . . . . 97
7.1.5. Methods of data collection . . . . . . . . . . . . . . . . . . . . . . . 97
7.1.6. Methods of data analysis . . . . . . . . . . . . . . . . . . . . . . . . 98
7.2. Case study results of CD pipeline A . . . . . . . . . . . . . . . . . . . . . . 98
7.2.1. Assessment of the security level of CD pipeline A by team members 99
7.2.2. Security objectives for CD pipeline A . . . . . . . . . . . . . . . . 99
7.2.3. Vulnerabilities detection and results . . . . . . . . . . . . . . . . . 100
7.2.4. Summary of results of CD pipeline A . . . . . . . . . . . . . . . . 104
7.3. Case study results of CD pipeline B . . . . . . . . . . . . . . . . . . . . . . 105
7.3.1. Assessment of the security level of CD pipeline B by team members 105
7.3.2. Security objectives for CD pipeline B . . . . . . . . . . . . . . . . 106
7.3.3. Vulnerabilities detection and results . . . . . . . . . . . . . . . . . 106
7.3.4. Summary of results of CD pipeline B . . . . . . . . . . . . . . . . 111
7.4. Proof of concept . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112
7.5. Discussion of both investigated CD pipeline results . . . . . . . . . . . . 113

8. Conclusion 115
8.1. Feedback of the project teams on the results . . . . . . . . . . . . . . . . 115
8.2. General applicability of the research results . . . . . . . . . . . . . . . . 116
8.3. Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
8.4. Lessons learned . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118
8.5. Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 119

A. Complete design and results of the survey 121

B. Insecure Jenkins configuration XML file 137

C. Results and used file excerpts of the case study 139

Bibliography 149

# List of Figures


2.1. Example continuous delivery pipeline . . . . . . . . . . . . . . . . . . . . . . 8
2.2. Example for the security attribute explanations . . . . . . . . . . . . . . . 11
2.3. Relations between security components . . . . . . . . . . . . . . . . . . . . . 14
2.4. The dependability and security tree . . . . . . . . . . . . . . . . . . . . . . 15
2.5. Example for the attack explanations . . . . . . . . . . . . . . . . . . . . . . 22
2.6. Build process and cross build injection attack . . . . . . . . . . . . . . . 24

3.1. Continuous deployment pipeline used in the paper of Ullah et al. . . . . 35

4.1. Survey question 6 results: Tools which are known and/or used . . . . . . 45
4.2. Survey question 7 results: The role of the developers if they interact with a CD pipeline. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
4.3. Survey question 8 results: The importance of the thesis topic in an industrial company . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
4.4. Survey question 9 results: The developer’s security involvement frequency during the development process . . . . . . . . . . . . . . . . . . . . . . . 47
4.5. Survey question 2 results: Assessment of the six security attributes through developers of a selected company . . . . . . . . . . . . . . . . . 49

5.1. Components and data flows of a generalized CD pipeline . . . . . . . . . 52
5.2. Jenkins user interface: global security configurations . . . . . . . . . . . 67

6.1. Output of the JobConfigHistory plugin . . . . . . . . . . . . . . . . . . . . 76
6.2. Docker Hub image security scanning results of official Jenkins image . . 80
6.3. Example results of Nexus Repository Manager Pro Health Check . . . . . 83
6.4. Insertion of OWASP ZAP Proxy in the CD process . . . . . . . . . . . . . 85

7.1. Structure of the investigated CD pipeline A . . . . . . . . . . . . . . . . . 93
7.2. Structure of the investigated CD pipeline B . . . . . . . . . . . . . . . . . 95
7.3. Pipeline A: Security assessment through team members . . . . . . . . . . 99
7.4. Pipeline A: Unencrypted remote repository connections . . . . . . . . . . 102

7.5. Pipeline B: Security assessment through team members . . . . . . . . . . . 106
7.6. Pipeline B: Results of Nexus Repository Health Check OSS . . . . . . . . . 110
7.7. Detection of vulnerabilities with OWSAP Zap Proxy . . . . . . . . . . . . 113

A.1. Survey question 2 results: Assessment of the six security attributes through developers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135
A.2. Survey question 1 results: Development experience of the participants . 135

B.1. Insecure Jenkins configuration XML . . . . . . . . . . . . . . . . . . . . . . 138

C.1. Pipeline A: OWASP dependency check results . . . . . . . . . . . . . . . . 140

# List of Tables


2.1. Definition of likelihood and impact levels . . . . . . . . . . . . . . . . . . . . 20
2.2. Overall risk severity matrix from OWASP . . . . . . . . . . . . . . . . . . . 20

5.1. STRIDE analysis: Tampering in the commit stage . . . . . . . . . . . . . . 54
5.2. STRIDE analysis: Information disclosure in the commit stage . . . . . . . 55
5.3. STRIDE analysis: Denial of service attack on the commit stage . . . . . . 55
5.4. STRIDE analysis: Tampering in the repository . . . . . . . . . . . . . . . 56
5.5. STRIDE analysis: Information disclosure in the repository . . . . . . . . 56
5.6. STRIDE analysis: Denial of service attack on the repository . . . . . . . . 57
5.7. STRIDE analysis: Tampering in the CI server . . . . . . . . . . . . . . . . 58
5.8. STRIDE analysis: Information disclosure in the CI server . . . . . . . . . 59
5.9. STRIDE analysis: Denial of service attack on the CI server . . . . . . . . 59
5.10. STRIDE analysis: Tampering in the build stage . . . . . . . . . . . . . . . 60
5.11. STRIDE analysis: Information disclosure in the build stage . . . . . . . . 61
5.12. STRIDE analysis: Denial of service attack on the build stage . . . . . . . 61
5.13. STRIDE analysis: Tampering in the artifact repository . . . . . . . . . . . 62
5.14. STRIDE analysis: Information disclosure in the artifact repository . . . . 62
5.15. STRIDE analysis: Denial of service attack on the artifact repository . . . 63
5.16. STRIDE analysis: Tampering in the testing stage . . . . . . . . . . . . . . 63
5.17. STRIDE analysis: Information disclosure in the testing stage . . . . . . . 64
5.18. STRIDE analysis: Denial of service attack on the testing stage . . . . . . 64
5.19. STRIDE analysis: Tampering in the production stage . . . . . . . . . . . 65
5.20. STRIDE analysis: Information disclosure in the production stage . . . . . 65
5.21. STRIDE analysis: Denial of service attack on the production stage . . . . 66

6.1. Overview: Tools which detect vulnerabilities in a single pipeline stage . . 88
6.2. Overview: Tools which detect vulnerabilities in multiple pipeline stages . 89

7.1. Pipeline A: Vulnerabilities in CD pipeline components . . . . . . . . . . . 101
7.2. Pipeline A: Summary of the detected overall risk severities . . . . . . . . 105

7.3. Pipeline B: Vulnerabilities in CD pipeline component . . . . . . . . . . . . 108
7.4. Pipeline B: Summary of the detected overall risk severities . . . . . . . . . 112

# List of Acronyms

**AAT** Automatic acceptance test

**API** Application programming interface

**AWS** Amazon Web Service

**CIA** Confidentiality, integrity and availability

**CD** Continuous delivery

**CI** Continuous integration

**CPE** Common Platform Enumeration

**CPS** Continuation-passing style

**CVE** Common Vulnerabilities and Exposures

**CVSS** Common Vulnerability Scoring System

**CWE** Common Weakness Enumeration

**DAST** Dynamic application security testing

**DevOps** Development and operations

**DFD** Data flow diagram

**DoS** Denial of service

**Amazon EC2** Web-Service Amazon Elastic Compute Cloud

**GDPR** General Data Protection Regulation

**GPG** GNU Privacy Guard

**GSN** Goal structuring notation

**HTTP** Hypertext Transfer Protocol

**HTTPS** Hypertext Transfer Protocol Secure

**JSON** JavaScript Object Notation

**LT** Load testing

**MITM** Man-In-The-Middle

**NIST** National Institute of Standards and Technology

**NSA** National Security Agency

**NVD** National Vulnerability Database

**OpenSCAP** Open Security Content Automation Protocol

**OWASP** Open Web Application Security Project

**REST** Representational state transfer

**RHC** Repository health check

**RHEL** Red Hat Enterprise Linux

**SAST** Static application security testing

**SCM** Software configuration management

**SDL** Software development lifecycle

**SFTP** Secure Shell File Transfer Protocol

**SSH** Secure Shell

**STIRDE** Spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege

**UI** User interface

**XSS** Cross-Site-Scripting

# List of Listings

2.1. Example Jenkinsfile of a declarative pipeline . . . . . . . . . . . . . . . . . 28

5.1. Exploit vulnerability on Jenkins configuration . . . . . . . . . . . . . . . . 67
5.2. Remove file with integrated Jenkins shell command executor . . . . . . . 68

6.1. Jenkins Pipeline Unit: Example unit test . . . . . . . . . . . . . . . . . . . 73

C.1. Unencrypted connection detection with OWASP Zap Proxy: gradle.properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
C.2. Unencrypted connection detection with OWASP Zap Proxy: Excerpt of build.gradle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139

Chapter 1

# Introduction

## 1.1. Motivation

According to the agile manifesto principle, it is "[the] highest priority [...] to satisfy the customer through early and continuous delivery of valuable software" [BBv+01]. By applying the continuous delivery (CD) approach companies are able to deploy application changes and features to the customer rapidly and reliably [HF11]. As reported by Humble and Farley [HF11], the continuous delivery approach is implemented by means of a CD pipeline. They mentioned that a CD pipeline consists of different stages. Several jobs are executed in each stage. If all jobs in each stage are successfully executed, then the next stage is triggered. If a job fails, the next stage is not performed and the execution of the CD pipeline is stopped.

According to Gilmore [Gil17], the results of the Hurwitz & Associates study show that 77% of the 150 participating IT decision makers have implemented the CD process in their projects. Based on these numbers one may assert with confidence that the CD process is an essential component in projects of companies.

Another approach which extends the CD process is called DevOps. To achieve the goal of continuous delivery a DevOps team deals with the development (dev) and operation (ops) aspects. The survey of Puppet and DORA [PD17], which was conducted in 2017, shows that 27% of the 3.200 participants 2017 work in a DevOps team. Compared to 2014, only 14% of the participants worked in such a team. The participants were employees of companies from various sectors of the economy such as finance, health, technology, and telecommunications. The results of the survey indicate a trend towards companies increasingly applying the DevOps approach.

Considering the fact that a new General Data Protection Regulation (GDPR) will enter into force in the European Union in May 2018, security aspects are currently in the minds of many companies [AS17]. This regulation demands that all companies are required to ensure at least a minimal level of security when processing personal data.

According to this regulation, data security has to included in software applications and their infrastructure by default. The companies are required to implement the following measures (GDPR Article 32[1]):

- Encryption of personal data
- Continuous safeguarding of the information security attributes (confidentiality, integrity and availability)
- Implement procedures which continuously execute security checks

As reported by Paulus [Pau12], insecure applications can severely damage the image of companies. Therefore, the aim of companies should be to secure their CD pipelines such that they have as few vulnerabilities as possible. Jimenez et al. [JMC09] mentioned that it is important to know just how secure applications are. This insight can be gained by means of vulnerability detection. In addition, they point out that it is necessary to know how the detected vulnerabilities can be eliminated. For this reason, it would make sense to implement the aforementioned measures of GDPR in CD pipelines, even if it is not legally necessary.

Feiman and McDonald [FM09] mentioned in 2009 that companies begin to understand the importance of security in applications. That is why there exists a demand for security tools on this account. However, Paulus [Pau12] has shown that in reality, as long as nothing happens, companies have little interest in application vulnerabilities and the resulting lack of security. Due to these facts, the following question arise:

**Which vulnerabilities are present in CD pipelines and how can they be detected?**

This thesis topic addresses the detection of vulnerabilities in CD pipelines in the DevOps context.

## 1.2. Research goal and research questions

The goal of this thesis is to detect software vulnerabilities in essential elements and in the structure of CD pipelines. Furthermore, the research results are applied to two industrially used CD pipelines in the context of a case study. The aim of this case study is to investigate the security of CD pipelines in the industry.

Through the thesis the following research questions are addressed:

[1]GDPR Article 32: https://gdpr-info.eu/art-32-gdpr/

$RQ_1$ Which vulnerability detection approaches and methods exist?

$RQ_2$ Which work was already conducted in this thesis field?

$RQ_3$ What knowledge do company's developers have in the area of securing and vulnerability detection of CD pipelines?

$RQ_4$ Which vulnerabilities exist with regards to CD pipelines?

$RQ_5$ Which tools are available to detect these identified vulnerabilities?

$RQ_6$ How secure are the selected company specific CD pipelines?

In order to answer these questions, the following research methods will be applied.

## 1.3. Research methods

In the second and third chapter the existing literature is researched to find basic concepts and related works. Similar works are those which obtain approaches, tools to secure CD pipelines or detect vulnerabilities in it. In addition, the internal structure of CD pipelines often corresponds to a number of web applications. Therefore, related works which are from the realm of general web security are also considered.

In the fourth chapter, a quantitative survey will give an overview of the participating company's developers familiarity with the field of securing and vulnerability detection of CD pipelines. An in-depth interview in form of an online survey was executed and 19 employees have participated.

The detection of vulnerabilities of a generalized CD pipeline is done with the threat modeling approach method STIRDE[2]. This method detects threats by categories. Each letter of the STRIDE name stands for a category. The vulnerabilities can be derived from the threats.

The results of the survey show that the CIA security attributes (confidentiality, integrity, and availability) are the most important ones in the view of the developers. Therefore, only the threat types TID[3] will be considered.

For automated vulnerability detection in CD pipelines, a literature research is conducted in the context DevOps and DevSecOps (Secure DevOps) to identify available tools. In addition, a further web search is carried out to identify tools which are capable of detecting the identified vulnerabilities.

In the case study, the investigation of two CD pipelines A and B which are used in

[2]STRIDE: spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege

[3]TID: tampering, information disclosure, denial of service

a selected company is performed. The aim of the case study is to detect vulnerabilities in instudrial used CD pipelines. For this research, some selected tools were installed and executed on these CD pipelines. The company can have an added value by using the selected tools. The identified vulnerabilities are mapped to a risk level which is based on the definition of the National Institute of Standards and Technology (NIST) and the Open Web Application Security Project (OWASP) risk rating methodology. The case study results should help the developers in the company to improve their CD pipelines.

#### Research results

The research in this thesis topic came to the conclusion that with the STRIDE method several vulnerabilities in a generalized CD pipeline can be derived. An unencrypted connection between two CD pipeline components and none access restrictions are two possible vulnerabilities which can occur in CD pipelines. In a survey, it was found out that the developers of a selected company are not familiar with this topic, but they see it as necessary to research it. The results of the survey show that these developers see the CIA security attributes as the important ones. Therefore, only those threats and vulnerabilities are considered which harm these attributes. In the next step, available tools were found out which can detect and mitigate the identified vulnerabilities in CD pipelines. For each CD pipeline stage, at least one tool exists to detect or mitigate vulnerabilities in CD pipelines. In the case study, two CD pipelines A and B were investigated. The investigation detected vulnerabilities which have a potentially high risk level. In most cases, a better level of security is not possible because the company's CD pipelines are installed on the customer's infrastructure. The results of the case study show that an unaware action of an employee can lead to an accidental exploitation of these detected vulnerabilities. Therefore, it is necessary that the company convinces the customer of the necessity of improving the security level of their CD pipeline. The aim should be to detect and remove vulnerabilities before they can be exploited. All in all, the CD pipeline is as secure as its weakest component.

## 1.4. Thesis structure

The thesis is structured as follows.

**Chapter 2 – Foundations** describes the basic concepts of the thesis which are necessary to understand the topic. The chapter includes known attacks which can be mapped on CD pipelines. Furthermore, it shows approaches and methods which exist to detect vulnerabilities in CD pipelines.

**Chapter 3 – Related work** presents the work which is related to the area of securing and detecting vulnerabilities in CD pipelines.

**Chapter 4 – Survey on the knowledge of developers** includes the description of the design, execution, and results of the survey. The survey is executed in a selected company to investigate the knowledge of developers on this thesis topic.

**Chapter 5 – Vulnerabilities in CD pipelines** shows the execution of the threat modeling approach method STRIDE on a generalized CD pipeline. This method is used to detect threats and vulnerabilities which can be present in existing CD pipelines.

**Chapter 6 – Tools for securing CD pipelines** is a collection of security tools which are able to detect the identified vulnerabilities. Moreover, it is presented if the tools are a vulnerability detection or mitigation tool.

**Chapter 7 – Case Study** presents the results of the execution of some selected tools on two industrial CD pipelines. The results show how secure the two industrial CD pipelines are.

**Chapter 8 – Conclusion** summarizes the results of the thesis. Additionally, it presents the feedback of the project teams of the investigated company on the found vulnerabilities. A general applicability of the research results is presented. In this chapter, lessons learned and an outlook on future works are shown.

Chapter 2

# Foundations

---

The following chapter outlines the foundations which are necessary to understand the thesis. This includes core aspects such as continuous delivery (Section 2.1) and CD pipelines (Section 2.2). In addition, the concept of DevOps (Section 2.3) and general security aspects (Section 2.4) are described. An essential part of this chapter is to find out which vulnerabilities exist (Section 2.5), which general attacks are present (Section 2.6). The last part of this section (Section 2.7) briefly describes the tools that are used in the survey (Chapter 4) and those which are used to construct the CD pipelines investigated in the case study (Chapter 7).

## 2.1. Continuous delivery (CD) and continuous deployment

As reported by Humble and Farley [HF11], CD is a process which "provides the ability to release new, working versions of [the] software several times a day. That means [the developer team] have to keep [their] application releasable at all times". The delivery or deployment process helps software development teams to reproduce occurring errors [HF11; Wol16]. Continuous deployment is, in contrary to CD, a fully-automated software delivery process. The only difference between the two processes is that the last step before the application is going into production is manually done in the CD process [Fow13; SAZ17]. If a company uses the CD process, it can decide when the software should be deployed [Fow13].

The CD and deployment process can be implemented as a pipeline [Fow13; HF11]. A code change leads to the fact that it goes automatically through different stages like build, test, integration and deployment. The automation of this process can be realized with various tools [RZB+15].

The goal of CD or continuous deployment is that all team members get feedback

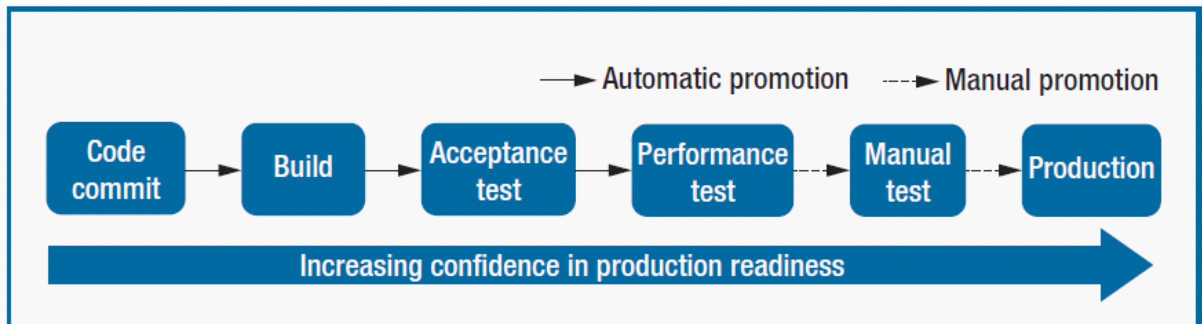



**Figure 2.1.:** Example continuous delivery pipeline adopted from [Che15]

on their code changes as early as possible [Che15; Fow06; HF11]. In addition, it enables a fast time-to-market which can increase user satisfaction [Che15].

## 2.2. CD pipeline

According to Humble and Farley [HF11], the CD pipeline is an automated process to get the software from the commit stage to the production stage. They mention that the implementation of CD pipelines depends on the company. The benefit of a pipeline is to get earlier feedback on each code change [Fow06; HF11]. Another advantage of an automated pipeline, named by Humble and Farley, is that failed stages and errors can be traced to causing code lines and to the developer who has programmed the errors. Every employee who has access rights can trigger the pipeline and create a new instance of it [HF11]. As a result of this a company needs no experts to deliver the software because every team member with pipeline access rights has the ability to deploy the software [HF11].

The structure of an example CD pipeline is shown in Figure 2.1. The following subsections briefly describe the individual stages of this basic CD pipeline as presented by Humble and Farley [HF11] and Chen [Che15].

#### Code commit

The first stage of the CD pipeline (see Figure 2.1) checks the code on the technical level. If developers commit a code change, a new instance of the pipeline is automatically triggered.

### Build

The next step "Build" (see Figure 2.1) performs continuous integration (CI) which builds a working application out of all artifacts. The execution of static code analysis, unit, and integration tests allows developers to determine which build is not suitable for the deployment to production. Furthermore, the developer gets insight about the status of the software health. As an example, this software health includes information about code style, code smells or code duplicates. If no errors are found at this stage the pipeline will automatically execute the next step. For further CD pipeline stages usages, all artifacts which are created are stored in an artifact repository.

### Acceptance test

In the CD pipeline the acceptance test stage (see Figure 2.1) is triggered if the build stage outputs that the software artifact has included no errors. On a functional and non-functional level, the software is tested against the specification. The results of these tests show whether the acceptance criteria are fulfilled. In this stage, the test execution takes more time than in the previous ones. For the test execution, a testing environment has to be set up. Test environment should correspond to the production environment as far as possible. The construction of the environment includes for example the creation and filling of databases, the setting up of external systems or the configuration of a server.

### Performance test

In former years, development teams typically executed performance tests at the end of the software development cycle. By establishing CD pipelines, it is possible to test the performance of the software with every code commit and every pass of the CD pipeline. As an example, performance tests are used to test the memory or CPU usage.

### Manual test

Although some extensive tests have been automatically performed up to this point, it is sometimes necessary to perform manual tests (e.g., user application tests performed by business users). Testers manually decide about the usability and the fulfilled requirements. For the manual test execution a testing environment is needed. Test environment should correspond to the production environment as far as possible. This environment

is automatically set up through the CD pipeline. If the test successfully passes this stage, the artifact is ready for going into production.

#### Production

In the release stage, the software is deployed to production. The developers whose CD pipelines were investigated in the case study mention that they do not have fully automated pipelines. This can be explained by the fact that, the customer wants to execute security or user tests on their own before the software is going into production. As a result, operators, testers or developers manually start the last step of the pipeline with a button click. The production step is not as potentially erroneous as in former times because the in used deployment scripts are tested in the stages before. A further advantage of the manual deployment step is that all team members and especially the operators can decide for themselves or in consultation with the customer which version and when it should be deployed.

## 2.3. DevOps

DevOps is a form of organization and extends CD. The approach combines the development (dev) and the IT operations (ops) in one team [DPL15; Wol16]. The main aspect of DevOps is the deeper collaboration between both groups. The CD pipeline needs knowledge of the operation and development team. All stakeholders (e.g., customer, developer, manager) can get fast feedback if the knowledge of both groups is present [Wol16]. In addition, the DevOps approach helps companies to achieve shorter delivery times without neglecting the quality of the software [DPL15].

## 2.4. Security

According to Paulus [Pau12], security is a software quality aspect. In addition, he mentions that in the context of security, it is necessary that software can withstand potentially destructive attacks. In his view software is secure if an attacker can not change the functionality of the software or manipulate the data.

According to Liggesmeyer [Lig09], two properties of security can be distinguished. Software which does not threaten the environment, persons or objects has the property of safety. Security is translated as data security. A secure application prevents the loss of information and unauthorized access to the application. A safe application does not

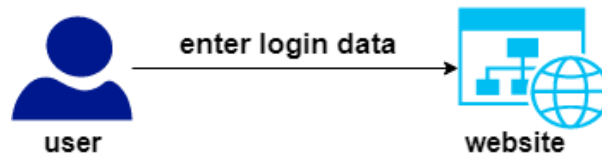



**Figure 2.2.:** Example for the security attribute explanations

harm the environment or persons.

To achieve this goal, security testing should be included at the earliest stage of the deployment process [JCL17]. The main objectives of software security are the aspects of information security (confidentiality, integrity, and availability) [Pau12]. Talukder and Chaitanya [TC08] mention other attributes which extend the attributes of information security. With respect to this thesis, the attributes Authentication and Authorization are relevant. These attributes belong to the persons who use this information. These attributes can be extended with a third one which is called nonrepudiation [Pes08].

### 2.4.1. Information security

"Information security ensures the confidentiality, [integrity, and availability] of information" [18]. Andress [And14] mentioned that these three aspects are called the CIA triad. If these three aspects are ensured, the security of the system can be guaranteed. The compliance of the CIA triad attributes should protect the data from being accessed by unauthorized persons and prevent the exploitation of vulnerabilities through attacks [18].
In the following subsections, the aforementioned six security attributes are briefly explained. An example for the security attribute explanations is shown in Figure 2.2. The scenario is that the user enters his user login data into a website. If this user logs in successfully, then he can see business-critical data on the website.

#### Confidentiality

This attribute ensures that only authorized persons gain access to business-critical data or information [18; Pau12]. One solution is to encrypt the transferred data that an unauthorized person cannot read it [TC08]. In the aforementioned example, confidentiality is given if the user logs in to the website and the login data is transferred in encrypted form so that an unauthorized person cannot read it when he intercepts the communication.

### Integrity

The goal of integrity is to protect the data and to ensure the accuracy and completeness of the data at any time [18; Pau12]. As a result of this, it should be ensured that an unauthorized person cannot manipulate data. In the aforementioned example, the integrity is given if user logs in to the website and no attacker can manipulate the messages of him. As an example, if an attacker changes the password in the user's request then the user will not be able to log in to the website.

### Availability

Availability ensures that the service/software is accessible when it is needed. Furthermore, it should be only available for a specific target group [18; Pau12; Pes08]. In the aforementioned example, availability is given if the user can log in to the website whenever he wants.

### Authentication

This approach validates the identity of the user [TC08]. The security attribute authentication ensures that the identity of the user is correct [Pes08]. In the aforementioned example, an authentication mechanism checks whether the user has entered a correct password to his username for the login.

### Authorization

Authorization manages the privileges of the user. According to this concept, it is necessary to ensure role-based security. Each user can have different privileges and rights to engage in activities on the software [TC08]. The security attribute ensures that a certain user can only perform activities which he has access rights or privileges for [Pes08]. The user in the aforementioned example has no admin rights, which means that if he logs in to the website, he cannot add or remove users.

### Nonrepudiation

The security attribute ensures that the user can not deny his interaction which the system [Pes08]. In the aforementioned example, the user logs in to the website and changes business-critical data. The security attribute nonrepudiation is given if the actions of the user are tracked and stored in a log file.

### 2.4.2. DevSecOps

The security extension of the DevOps approach is called DevSecOps or SecDevOps (Secure DevOps) [Fal16; MO16; UW16]. This approach is the result of the increasing security requirements in the software life cycle [Fal16]. As a consequence of this, security testing should be applied at an early stage in the software development and CD process. Through the earlier detection of errors, the fixing costs are lower [Car17; Fal16]. On the one side, the DevSecOps approach can improve security in the system, but on the other side there is a risk that the rapid delivery process overlooks the security aspects [UW16].

## 2.5. Vulnerabilities

The first part of this section describes the nature of vulnerabilities. Furthermore, a definition is given on the relationship between vulnerabilities, risks, threats and assets. The next subsection shows the attributes, their threats, as well as the means for attaining dependability and security which were published by Avizienis et al. [ALRL04]. In addition, it is essential to know what software and pipeline vulnerabilities are. To detect vulnerabilities in the CD pipeline structure and its components, the Open Web Application Security Project (OWASP) and software weakness lists which publish the existed vulnerabilities should be known. The last part of this section describes existing approaches which can detect vulnerabilities.

### 2.5.1. Vulnerabilities and the relation to threat, asset and risk

The NIST defines **vulnerability** as a “Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source” [FI13]. Vulnerabilities can occur anywhere in a computer or software program [TC08].

If vulnerabilities are considered, then it is necessary to look at the related components. According to Farn et al. [FLF04], vulnerabilities are connected with threats, assets, values and risks. The Figure 2.3 shows the relation between those components. In the following these components and their relations are described.

Based on Eckert [Eck13], a **threat** pursues the goal of exploiting a vulnerability and compromising security attributes. Additionally, the exploitation of the vulnerability by the threat can lead to unauthorized extraction of **assets**. The amount of damage depends on the gained type of data. As reported by Eckert, assets are the sources which should be protected. Each asset has a **value**. The value indicates the effort or the money

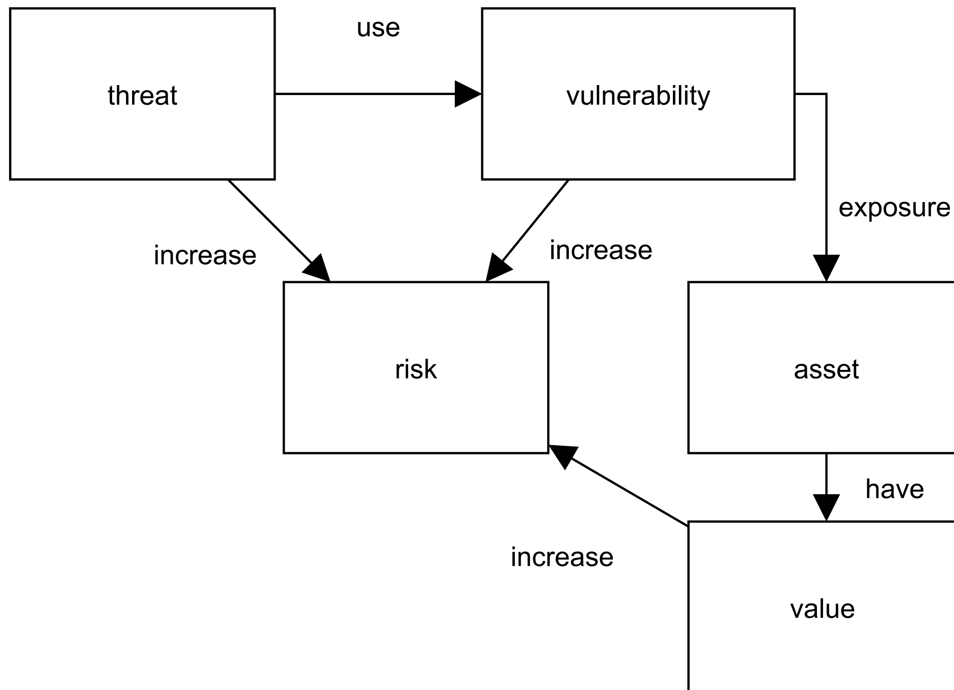



**Figure 2.3.:** Relations between security components based on [FLF04]

a company has to put in if assets are damaged or lost by an attack [OWAa]. The potential commercial impact is described through the value [FLF04]. The **risk** is the product of the probability of the threat occurrence, the vulnerability exposure, and the value of an asset [OWAa].

### 2.5.2. Threats, attributes, and means for attaining dependability and security

According to Avizienis et al. [ALRL04], the CIA triad security attributes are a part of the dependability attributes. They mentioned that “dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable” [ALRL04]. In addition to the CIA triad attributes, there are three other dependability attributes. These are “reliability (continuity of correct service), safety (absence of catastrophic consequences on the user(s) and the environment), and maintainability (ability to undergo modifications, and repairs)” [ALRL04]. Avizienis et al. presented that there are three categories of threats that can occur in the context of dependability and security. These three categories are failure, error and fault. A failure exists if the software state deviates from the correct state (functional specification). This deviation is called

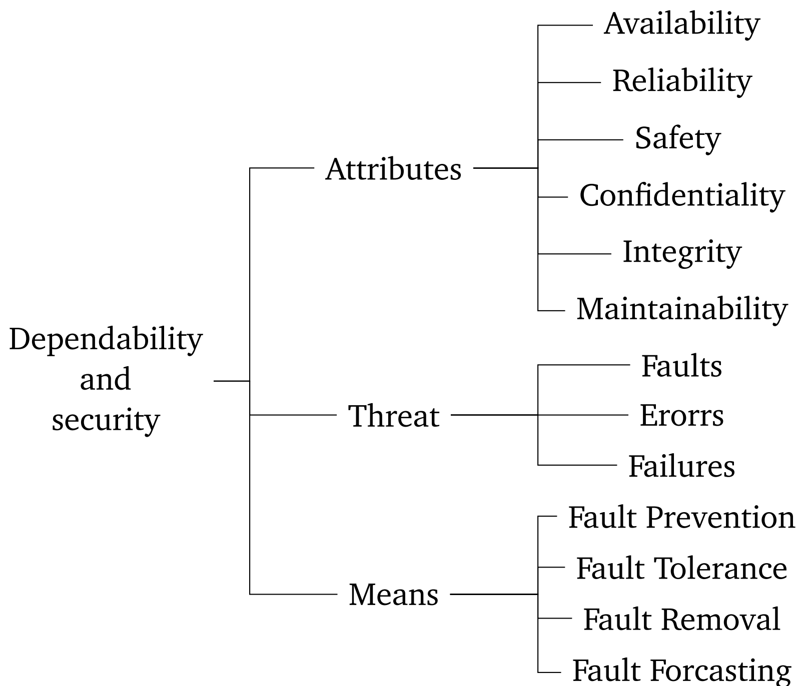



**Figure 2.4.:** The dependability and security tree based on [ALRL04]

an error. The cause of an error is called fault which can, for example, be a software, accidental, human-made, malicious or non-malicious fault. Avizienis et al. presented four means to achieve the attributes of dependability and security: "prevention (prevent an occurrence of fault), fault tolerance (avoid service failures), fault removal (reduce severity faults), fault forecasting (estimate number and consequences of faults)". An overview of all attributes, threats, and means which were mentioned by Avizienis et al. is shown Figure 2.4.

### 2.5.3. Software vulnerabilities

As mentioned above, software vulnerabilities are flaws in the systems. According to Jimenez et al. [JMC09], programming mistakes and errors in the software are one possibility which can lead to vulnerabilities. Due to the increasing amount of software systems there exist more vulnerabilities [JMC09]. One possible aim of an attacker is to get sensitive data or gain the control over the system [BM05; JMC09]. As reported by Jimenez et al., vulnerabilities are the interfaces between a system and an attacker. If an attacker gains the access rights, he can damage the system. Furthermore, Jimenez et al. mention that if there exists vulnerabilities the attacker can inject malicious code or manipulate the data of a system. Therefore, it is necessary to know where the

vulnerabilities are, but it is impossible to find all of them. It is only possible to show the existence of vulnerabilities and not of their absence [JMC09].

### 2.5.4. Pipeline vulnerabilities

Pipeline vulnerabilities are leaks in the delivery processes. If a company overlooks security aspects, then there will likely vulnerabilities in each of the separate delivery stages. Following the paper of Gruhn et al. [GHJ13], the CI system tools have more vulnerabilities than it is communicated through the vulnerability databases or communities. One attack vector is the execution of the code through the build jobs. Attackers can include malicious code at specific places in the CD pipeline. During the build process the code is executed as arbitrary code [GHJ13].

### 2.5.5. Open Web Application Security Project (OWASP) and software weakness lists

There exist an organization [OWA17a] and several databases which present known vulnerabilities [LS04]. The organization and the databases are described in the following section.

#### Open Web Application Security Project (OWASP)

OWASP [OWA17c] is an international organization and open community for companies, organizations and all individuals who are concerned with software risks and vulnerabilities. OWASP provides tools, documentation and videos for free which developers can use to secure their applications. . The organization publishes the OWASP Top 10 in regular time steps. The list includes the most critical web application security risks. Since a pipeline can consist of different web applications, these vulnerabilities can also occur there. In November 2017 the OWASP Top 2017 list [OWA17a] was published. The top three web security risks 2017 are injections, broken authentication, and sensitive data exposure. Attackers can use the injection vulnerability to inject queries to manipulate data in the databases or to get access to sensitive data. User inputs are the main weak points where this vulnerability occurs. The OWASP Top 10 list should help developers to develop secure web applications [OWA17a].

#### Common Weakness Enumeration (CWE)

"CWE is a community-developed list of common software security weaknesses" [MIT]. The community includes major software companies such as Apple and the Microsoft Corporation. The CWE list has approximately 750 entries [MIT].

#### Common Vulnerabilities and Exposures (CVE)

The CVE list [CVE] includes cybersecurity vulnerabilities. The list has over 90.000 entries. This list is included in the National Vulnerability Database (NVD) [Nat].

#### VulDB

VulDB is the number one worldwide of the vulnerability databases [SCI]. Since 1970, the experts of this community have been documenting the upcoming vulnerabilities on a daily basis. The database contains more than 110,000 entries [SCI]. This database includes for example the entries of CWE and CVE. Since 2006, an average of 15 to 44 vulnerabilities have occurred per day [SCI]. In contrary to the other databases, VulDB describes, categorizes and adds information to the vulnerability. Additionally, the vulnerability is evaluated with the Common Vulnerability Scoring System (CVSS) score and countermeasures are published [SCI]. The CVSS is "an open standard for risk metrics of security issues" [SCI]. This database gives developers an overview of the applications and their existing vulnerabilities. It also shows the developer how to mitigate or remove the vulnerability [SCI].

### 2.5.6. Vulnerability detection approaches

Several approaches are known for the detection of vulnerabilities in applications. The first part of this section describes the threat modeling approach. The threat modeling approach can be used to detect the vulnerabilities in applications because a threat can exploit the vulnerability to gain access to assets (see Figure 2.3). As a result of this, vulnerabilities can be derived from threats. The second part of this section includes other detection approaches such as static application security testing (SAST), dynamic application security testing (DAST), and penetration testing.

### Threat modeling

Threat modeling is an approach to detect potential vulnerabilities, further risks or threats in an application as early as possible [MLY05; OWA17b]. Threat modeling extracts the components of the system and considers the possible entry points from an attacker's point of view [OWA17b]. The aim is to find the threats and errors in an application as early as possible and to reduce the error fixing costs [Sho14b]. An error or rather a vulnerability which is found in the production stage is 30 times more expensive to resolve than it is detected in the first stage [Tas02]. Therefore, it is reasonable to do threat modeling [MLY05; Tas02]. This approach is an essential method for web application [OWA17b] or network systems [Tas02]. According to Hussain et al. [HKA+14], various threat modeling approaches were developed over the time. As reported by OWASP [OWA17b], threat modeling is a process and can be executed in the three following steps:

1. Decompose the application:
   The first step is to consider the application in detail. In this step, the application has to be understood with all its components and connections [MLY05; OWA17b]. In this phase, it is necessary to identify the application's adversaries and their possible entry points [MLY05]. The application is visualized in a data flow diagram (DFD).

2. Determine and rank threats:
   The second step is to identify the threats in the application. According to Hussian et al. [HKA+14], the STRIDE threat modeling method "is the most widely used threat method". Therefore, the thesis focuses on this method. Threats can be sorted into different categories according to STRIDE. Each character of this name stands for one threat category. Every exploitation of a vulnerability is an offense against the security attributes of the information security (see Section 2.4.1). The six categories of STRIDE are the following [Sho14b]:

   - **S**poofing: The attacker pretends to be somebody who he is not (Section 2.4.1 — hurts authentication)
   - **T**ampering: The attacker modifies (sensible) data for which he has no authorization.(Section 2.4.1 — hurts integrity)
   - **R**epudiation: The attacker can deny his activities (Section 2.4.1 — hurts nonrepudiation)
   - **I**nformation disclosure: The software/application displays information to unauthorized persons/attackers (Section 2.4.1 — hurts confidentiality)

- **D**enial of service: The attacker slows down the application, causes it to crash or fills the memory (Section 2.4.1 — hurts availability)
- **E**levation of privilege: Program or user can do things for which they have insufficient access rights (Section 2.4.1 — hurts authorization)

Out of the threats the vulnerabilities can be derived.

3. Determine countermeasures and mitigation:
If threats and vulnerabilities are detected, it is necessary to determine the risk. This can help to prioritize which threat or vulnerability should be mitigated or removed. The Microsoft threat-risk ranking model (DREAD) is one method. DREAD stands for [OWA17b]:
    - **D**amage: extent of the attack
    - **R**eproducibility: effort to repeat the attack
    - **E**xploitability: effort to exploit the attack
    - **A**ffected User: number of users who are affected by the attack
    - **D**iscoverability: difficulty to find vulnerability

The risk is calculated from the sum of the five numbers divided by five at the end. If vulnerabilities are detected the stakeholders have to decide how to mitigate them. The team should find out if there exists countermeasures to reduce the vulnerability. It is possible that not all vulnerabilities can be mitigated [OWA17b]. The OWASP risk rating methodology [OWAb] is another approach to show the DevOps team the overall risk severity. The severity is the product of likelihood and impact (see Equation (2.1)).

$$OverallRiskSeverity = Likelihood * Impact \quad (2.1)$$

Based on this methodology, Maković [Mar] developed a calculator that determines the overall risk severity. This calculation is time-consuming to carry out and cannot be determined so precisely since not all required data for this purpose can be determined exactly in the case study pipelines. In this thesis the risk levels are defined based on this calculator selection options and the definition of the NIST Special Publication 800-30 [SGF02]. Table 2.1 shows how the likelihood and impact factors are defined. These two factors can have three security levels: low, medium, and high. A higher likelihood factor is given if the attacker needs no resource access rights to exploit the vulnerability. In addition, he needs no or minor knowledge about the system. If the attacker needs full access rights and a high system knowledge then a low likelihood factor exists because it is improbable that an attacker can exploit the vulnerability. The damage to the company and

| | **Likelihood definition** |
|---|---|
| Low | •Need high system knowledge<br>•Require full resource access rights |
| Medium | •Need system knowledge<br>•Require some resource access rights |
| High | •Need none or minor system knowledge<br>•Require no resource access rights |
| | **Impact definition** |
| Low | •Loss of data or resources<br>•No or small damage to the company's image |
| Medium | •Expensive loss of significant data or resources<br>•Damage to the company's image |
| High | •Very expensive loss of significant data or resources<br>•Major damage to the company's image |

**Table 2.1.:** Definition of likelihood and impact levels based on [Mar; SGF02]

| | | Impact | | |
|---|---|---|---|---|
| | | Low | Medium | High |
| Likelihood | Low | None | Low | Medium |
| | Medium | Low | Medium | High |
| | High | Medium | High | Critical |

**Table 2.2.:** Overall risk severity matrix from OWASP based on [OWAb]

the amount of data which is gained through the exploitation of the vulnerability defines the impact factor. On the contrary, the impact level is small if the attacker gains data, but the caused damage has no major consequences for the company. Table 2.2 shows the matrix which is used to calculate the overall risk severity. This matrix was created by OWASP [OWAb]. The two levels of the likelihood and the impact can be deduced in the matrix. The resulting cell expresses the overall risk severity of this vulnerability. The overall risk severity can have the following levels: note, low, medium, high and critical. If the level is note then no risk is present. If the calculation reveals that the risk is critical then the company should act as quickly as possible to prevent further damage. This matrix can be used to show the development and operation team the risk level of the detected vulnerabilities and the security status of their application.

### Static application security testing (SAST)

SAST tools use static analysis methods to detect software vulnerabilities in source and bytecode. The static analysis method is a software testing approach which detects formal errors in the source code. The SAST tools are applied before the runtime of the application and at the beginning of the software lifecycle. These tools analyze the application "from the inside out" in the development, testing, and build phase [FM09].

### Dynamic application security testing (DAST)

In contrary to the SAST tools, the DAST tools dynamically detect the vulnerabilities at runtime. These tools analyze the application behavior from the "outside" for example a through black box testing [Gue15].

### Penetration test

With this form of test, a project team can automate the execution of attacks on the application. Penetration tests are not used to test functions. The goal of this method is to detect the back doors or flaws in the application before attackers can exploit them [Bir16]. The disadvantage of this method is that security experts are needed. The effort and costs to execute these tests are high [Gue15]

## 2.6. Attacks

If a system has vulnerabilities and an attacker can exploit them, then it is called an attack [TC08]. The way an attacker exploits the software vulnerabilities are categorized for example by Langweg and Snekkenes [LS04] into three parts. The first category is the location. It is important to know where the attacker exploits the vulnerability. Exemplary locations are the user input, operating system or other applications. The second category of the attacks is called the cause. If there exists programming flaw then the attacker can exploit them. A programming flaw can be an incorrect validation of the input data during a website login. If an incorrect validation of input data exists then it is possible for an attacker to inject malicious code into the application. In this category, the attacker exploits programming and validation mistakes. The third category is called the impact. All attacks in this category cause a failure of the software. The attacks on the vulnerabilities can lead to the execution of arbitrary code, denial of service or a change of the resources [LS04].

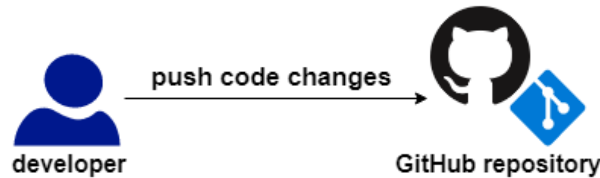



**Figure 2.5.:** Example for the attack explanations

Barzin [Bar07] mentions that attacks are dependent on the software engineering process phases. Attacks can be made on the architecture and design of the software. In addition, during the software operation further attacks arise [Bar07]. According to Talukder and Chaitanya [TC08], there also exist authentication attacks.

In the last subsection attacks on CD pipeline stages and components are briefly described. Of the total number of attacks, only those which can be relevant to the thesis are described in the following section.

In the following subsections, the aforementioned attack categories are explained. An example for the attack explanations is shown in Figure 2.5. The scenario is that a developer wants to push code changes to the GitHub[1] repository. GitHub is a development platform on which software code can be hosted and reviewed. Before the developer can perform this push, a commit of the code changes has to be done. With a push the code changes are uploaded in the repository.

## 2.6.1. Attacks based on architecture and design

Attacks which exploit vulnerabilities in the architecture and design of the software have several aims. On the one side, there are passive attacks. During these attacks the attacker can read sensitive data like username and password or tracks the traffic of the network [Bar07; TC08]. On the other side, it can be an active attack by which the attacker disturbs the system or manipulates data [TC08].

#### Man-In-The-Middle (MITM) attack

As all things are connected via the Internet according to Conti et al. [CDL16], it is necessary to prevent the MITM attack. Conti et al. mention that in the area of computer security this attack is very well-known. The aim of this attack is that the attacker gains control over the connection between two endpoints [CDL16]. The attacker wants to

[1]https://github.com/

manipulate the transfered data without any rights (authorization) and authentication steps [Bar07; CDL16]. If the attacker is successful, then he is the MITM between the transmitter and the receiver. The attacker manipulates both sides into believing that the connection is correct [Bar07]. Conti et al. mention that it depends on the used technique, the communication channel and the location of the attacker how the attack is executed and what effects it has on the two endpoints. To summarize up, the MITM attack is both an active and passive attack. A successful attack enables the attacker to gain access on transferred data and in addition manipulate them.

In the aforementioned example, a passive attack exists if an attacker can intercept the connection between the developer and the GitHub server as MITM and read the transferred messages or the code changes. In the case of an active attack, the attacker as MITM can manipulate the committed messages or the code.

### 2.6.2. Attacks during operation

If an application is in use or components are running then the aim of an attacker is to gain access to the systems or shut down/crash the application/components. The denial of service (DoS) attack and the default-account attack are two possible attacks which can be executed during the application operation.

#### Denial of service (DoS) attack

Through a DoS attack, the attacker can make the system unusable. The aim of the attacker is to execute arbitrary code or to manipulate the system such that it slows down or crashes. Another possible target of the attacker is that a website is no longer available for the user. Attacks of this category are for example viruses or worms (self-propagated viruses) [TC08].

In the aforementioned example, a DoS attack would be if an attacker can slow or shut down the GitHub server. If the server is down then the developer cannot push any code changes.

#### Default-account attack

The default-account attack uses the situation that applications often deploy standard, default or preassigned user accounts. In the most cases, those accounts are never changed. This unaware action allows the attacker to easily access the rights and account of the application administrator [Bar07].

In the aforementioned example, a default attack could occur if the GitHub server

is delivered with default login data (for example username: admin, password: admin) and this developer team has never changed this data. As a result of this, an attacker can guess the login data and logs in to the GitHub repository.

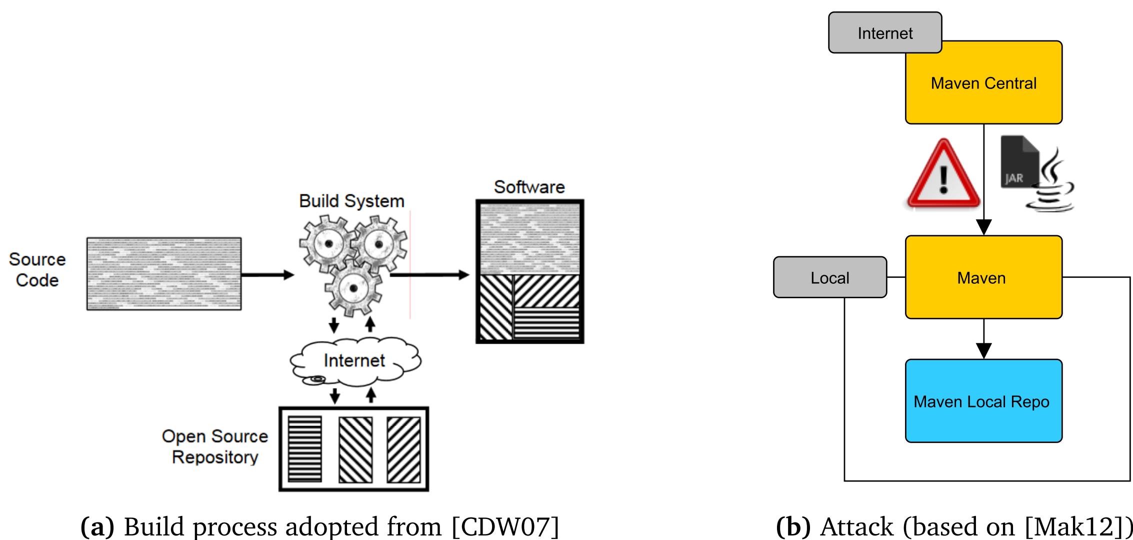
### 2.6.3. Attacks based on authorization

According to Talukder and Chaitanya [TC08], authorization is the first level to ensure security. If there exist known vulnerabilities in this area, then an attacker can gain access to the application without user credentials. They mention that the authorized user offers the identity of himself when he logs in to an application. If an attacker can obtain user login data, then he has the same rights as the users.

In the aforementioned example, this attack can occur if an attacker gets the login data of the developer. As a result of this, he can manipulate the CD pipeline scripts or the developed application.

### 2.6.4. Attacks on CD pipeline stages and components

There exist specific attacks which can be mapped on CD pipelines. These attacks are called cross build injection attack and compromise CD pipeline components.




**(a)** Build process adopted from [CDW07]  **(b)** Attack (based on [Mak12])

**Figure 2.6.:** Build process and cross build injection attack

### Cross build injection attack

This attack allows attackers to use the Internet connection to inject malicious code or manipulate transferred data. If an attacker gets successfully access of the application, then the build process can execute the injected code [CDW07]. The entry points of an attacker in the build process are the connections to external sources. Figure 2.6a illustrates that the build system can have connections over the Internet to an open source repository [CDW07]. The attacker intercepts the connection as MITM and captures or manipulates the data.

As reported by Mak [Mak12], most software uses external libraries. He points out that the use of external sources means that a developer team have to trust strangers and their developed code. One known possible attack is shown in Figure 2.6b. There exists one external repository (Maven Central) and one local repository. If the Java code is built with Maven[2] then the external libraries are downloaded from the Maven Central and stored in the local one. The software is built with these downloaded libraries. If the connection to the external repository is an unencrypted Hypertext Transfer Protocol (HTTP) connection then the possible effect can be that the attacker replaces the transferred libraries and inject malicious code into the build process [Mak12].

### Compromise CD pipeline components

According to the company Trend Micro [Tre18], today servers are often misconfigured and insecure. In most cases, companies do not know that their data is accessible to the public. Tesla[3] has fallen victim to an cryptojacking attack that compromised internal test car data on their Amazon Web Service (AWS)[4] cloud resources [Tre18]. In addition to that, the attackers use the resources to install their cryptojacking operations. For the cryptojacking attack, an attacker has to installed a miner software on the resources [Hay18]. After that, the attacked resources can be used to create cryptocurrency. The cause of this attack was a vulnerability in the Kubernetes[5] console. Tesla was lucky that no customer or the vehicle safety was compromised.

Jenkins[6] servers are also increasingly affected by cryptojacking attacks [Tre18]. An attacker has the possibility to exploit the in April 2017 published vulnerability[7] in Jenkins. For this attack, the attacker has to send requests to Jenkins. With the first

[2] https://maven.apache.org/
[3] https://www.tesla.com
[4] https://aws.amazon.com/
[5] https://kubernetes.io/
[6] https://jenkins.io/
[7] Jenkins vulnerability: CVE-2017-1000353

request an object tries to compromise the Jenkins server. If this action is successful then a JenkinsMiner is installed on it. This attack can slow down the Jenkins server or cause a DoS attack [Tre18].

## 2.7. DevSecOps and static analysis tools

DevSecOps tools help to automate the CD process. Both commercial and open source tools exist. The tools which are constructing the investigated CD pipelines in Section 7.1.3 are described below. In addition to that, further tools are described which are used in the survey (see Section 4.2). The tools selected for the survey are either used by the employees of the investigated company or are known to the researcher up to this point in time.

### Code analysis tools

Before a developer commits code changes, the source code should be checked for programming flaws. There are several tools which are able to do that. **Checkstyle**[8] is a tool which checks the coding standard of the programming language Java. Another tool is **PMD**[9] which detects common programming flaws in source code. In contrary to Checkstyle, it supports different programming languages. **SonarQube**[10] is another tool to detect bugs, code smells (source code which is not easily maintained) and security vulnerabilities in the source code. A feature of SonarQube is to show the health of an application. A tool which is able to detect bugs in the Java source code is called **FindBugs**[11]. As an example, there are bugs found in the categories bad practice, correctness and performance.

### Software configuration management (SCM) tools

Software configuration management tools are used to version and track the source code after a commit. Those tools give an overview over the source code and its changes.

**Bitbucket** [Atl] from Atlassian is a user interface for the version control system git[12]. This system supports collaborative teamwork development. It helps developers to

[8]http://checkstyle.sourceforge.net/
[9]https://pmd.github.io/
[10]https://www.sonarqube.org/
[11]http://findbugs.sourceforge.net/
[12]https://git-scm.com/

track the status and the whole development history of the software. Through version control systems the developers can control their code changes and the source code. Programmed mistakes and code changes can be traced to developers. An additional feature is that the created repositories can be connected with a CI system.

There exists a similar tool to Bitbucket which is called **GitLab** [Git]. GitLab has similar features as Bitbucket. One function is to track the software development progress or it can be used as version control system.

**GitHub**[13] is a development platform on which software code can be hosted and reviewed. A further feature of GitHub is that projects can be managed.

### CI tools

CI tools are used to construct the CD pipeline. In addition, the stages of the CD pipeline can be monitored and triggered.

**Jenkins** [Jena] is an open-source CI server. The server has the functionality to automate the tasks of build, test, and deployment. Jenkins can be used for all stages of the CD pipeline.

The structure of a CD pipeline is deposited in a Jenkinsfile [Jenb]. The structure of a sample declarative pipeline is shown in Listing 2.1. This pipeline shows the necessary parts for creating a CD pipeline. The agent declares the workspace in which the pipeline is executed [Jenb]. Every pipeline needs such declaration. The stage defines the specific parts of the pipeline. The build step is not the replacement for the build tools and has to include one of them, for example Maven[14] or Gradle[15]. Optional in the test stage the failures and successes of the tests are recorded and visualized through the Jenkins web user interface. The deployment stage is triggered if the build and the test stages are executed successfully. The artifacts which are built in the previous stages can be published in an artifact repository or into production [Jena]. Jenkins can be extended with several plugins to automate every stage of the pipeline and as an example, to establish a connection to other tools or clouds [Jena].

Further top three CI tools are [Pec16]: **Travis CI**[16], **GoCD**[17], **TeamCity**[18]. TeamCity is a commercial tool. One newer open source CI tool is **Councourse CI** [Con]. From the beginning this tool deals with the problems of Jenkins. Jenkins, for example, does not support pipelines first. In contrary to Jenkins, Concourse CI is not a construction of

[13]https://github.com/
[14]https://maven.apache.org/
[15]https://gradle.org/
[16]https://travis-ci.org/
[17]https://www.gocd.org/
[18]https://www.jetbrains.com/teamcity/

**Listing 2.1** Example Jenkinsfile of a declarative pipeline based on [Jena]

```
1  pipeline {
2        agent any
3
4        stages {
5              stage('Build') {
6                    steps {
7                          echo 'Building..'
8                    }
9              }
10             stage('Test') {
11                   steps {
12                         echo 'Testing..'
13                   }
14             }
15             stage('Deploy') {
16                   steps {
17                         echo 'Deploying....'
18                   }
19             }
20       }
21 }
```

several plugins. The aim of Concourse CI is to make the pipeline understandable despite the complexity of the project [Con].

### Containerization tools

**Docker** [Doc] is a software container platform. With this platform, software applications can be separated from their infrastructure. The Docker platform isolates the environment into containers. Applications can be developed and tested in containers. To create a Docker container, the developer has to build a Docker image. Images are declared by a Dockerfile that contains the steps for creating the image. An executable instance of an image is called container. Other components such as the Docker engine (server), the Docker client which is used to interact with the user and the Docker registry are a part of the Docker family. The Docker registry is an application which can store Docker images. In the CI context, Docker containers can be used to set up multiple build slaves [Doc].

**Kubernetes**[19] is an orchestrator for containerized applications. With this tool those applications can be scaled, managed and deployed.

[19]https://kubernetes.io/

### Cloud tools and environment server

The **Web-Service Amazon Elastic Compute Cloud (Amazon EC2)** [AWSb] is a service which provides secure and elastic computing power in the cloud. With a graphical user interface the developer can add more capacity if it is necessary. The customer only has to pay for the used capacity (pay per use).

With **Amazon EC2 container service** [AWSa] developers can deploy different applications in the Amazon Web Service (AWS) cloud. In this case, the Docker containers run on clusters. This service allows to monitor the application. Furthermore, it is possible to run and rollback the deployment of the Docker containers.

The **IBM WebSphere Application Server 8** [IBM] is mainly a Java EE 6 application server[20] on which the developed application runs. In the investigated pipeline B (see Chapter 7) it is used as test server.

### Config and provisioning tools

**Puppet** [Pup] is an open source DevOps platform which can be used to automate infrastructure. The platform helps companies to get an overview on their infrastructure. The functions of Puppet are: automate and monitor the infrastructure and fast delivery of the software. The description of the infrastructure as a code is possible through Puppet. A result of this is that the infrastructure code can be tested. Changes on the infrastructure can be done easily, securely, and reliably. The language of Puppet is Ruby[21]. Tools similar to Puppet are **Chef**[22] or **Ansible**[23].

**Consul** [Has] of Hashicorp is a configuration management tool. In the investigated pipeline B the key-value store option is used. The key-value store is a non relational database in which each value is assigned to a key. Each step of the pipeline is stored there.

### Repository management tools

**JFrog Artifactory** [JFr] is an artifact repository manager on the market. In a sufficient busy project every day several builds are started and as a result many artifacts are created. This repository provides an overview of the artifacts and a version control

[20]http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html
[21]https://www.ruby-lang.org/en/
[22]https://www.chef.io/chef/
[23]https://www.ansible.com/

repository of the generated artifacts. JFrog can be connected to CI tools. As an example, Jenkins and JFrog can be integrated. In addition, JFrog Artifactory can be connected to remote repositories such as Maven Central.

Another artifact repository is the **Nexus Repository** of Sonatype [Son]. In contrary to JFrog Artifactory, the Nexus Repository includes a repository health check (Repository health check (RHC)) which checks the components of applications. The result of this check shows components for which vulnerabilities are detected. In addition to that, the top five vulnerable components are displayed. This ranking is calculated out of the impact and the severity of the vulnerability. Furthermore, the tool gives the developer indications how to mitigate the vulnerability. The Nexus Repository is also compatible with Jenkins.

#### Deployment tools

**Rundeck** [Run] is a server that can be used for deployment. One part of the server's functionality is that operations can automatically or manually deploy the created nodes. The nodes contain specific deployment scripts.

With the open-source **Spinnaker**[24] platform of Netflix applications can be deployed on several clouds such as Amazon EC2 or Google. For the deployment, images are created and provided.

**Hockey App** [Mic] is a platform for apps (e.g., iOS, android, windows). If an application is a mobile app, it is possible to deploy it in the production stage on the HockeyApp platform. The user can download and use the apps from this store.

In the investigated pipeline B, a **SFTP server** is used as an intermediate storage for artifacts. The server is for example from the OpenSSH Project[25] which is available in every Linux Distribution. The transfer protocol is the Secure Shell File Transfer Protocol (SFTP)[26]. This protocol can be used to securely transfer the artifact from a build server to a deployment server. If an artifact is deployed on this server then it is in production and can be installed through the customer.

#### Security Tools

FindBugs can be extended through the plugin **FindBugsSecurity**[27]. This allows that the source code of Java-based and Android applications can be inspected for vulnera-

[24] https://www.spinnaker.io/
[25] https://www.openssh.com/
[26] https://tools.ietf.org/html/draft-ietf-secsh-filexfer-13
[27] http://findbugs.sourceforge.net/

bilities. There are other tools such as dependency tools for example **Snyk**[28] or **Black Duck Hub**[29] which detects vulnerabilities in open source components and containers. Netflix publishes the open-source security tool **Security Monkey**[30] which monitors AWS accounts for insecure configurations. Another tool is **JFrog Xray** which is the extension to the JFrog Artifactory tool [JFr]. JFrog Xray does an universal analysis of artifacts which are stored in the JFrog Artifactory. **BDD-Security**[31] is a security testing framework which can test security requirements.

This is only a short extract of the security tools. They are mentioned here because they are used in the survey. Further details about security tools are given in Chapter 6.

[28] https://snyk.io/
[29] https://www.blackducksoftware.com/
[30] https://github.com/Netflix/security_monkey
[31] https://www.continuumsecurity.net/bdd-security/

Chapter 3

# Related work

---

The related work section shows the state of research in this thesis topic. Pipelines can consist out of various web based applications (CD pipeline tools). The first part of this section considers papers which have dealt with securing web applications (Section 3.1). The second part is about papers which have engaged in securing pipelines (Section 3.2). In addition to that, papers are listed which are based on the threat modeling approach to detect threats and vulnerabilities (Section 3.3). These papers show that threat modeling can be used to detect threats and vulnerabilities in CD pipelines. One focus of the thesis is to find out which tools are available to detect vulnerabilities in CD pipelines. Papers containing security techniques and security tools that extend the DevOps approach complete this chapter (Section 3.4). The papers in which the tools are listed form the basis for Chapter 6.

## 3.1. Securing web applications

A CD pipeline consists of tools that are in most cases web based applications (e.g., Jenkins, Bitbucket, JFrog Artifactory). Deepa and Thilagam [DT16] created a collection of known approaches that detect vulnerabilities in web applications. In addition, they listed analytical techniques and how developers can prevent vulnerabilities in web applications. They mentioned that vulnerabilities are present in all phases of the software development lifecycle (SDL), so it is important that they are investigated in every phase of the lifecycle. To secure web applications, the first step for developers is to develop secure program code [DT16]. Developers should follow guidelines and use, for example, programming languages which automatically do datatype checking and memory management (garbage collection).

Vulnerabilities occur mostly through mistakes in the source code [LWC+12]. Lee et al. [LWC+12] discovered that vulnerabilities in web applications and servers can be

detected with the fuzzing method. To use this method abuse cases which are created out of known vulnerabilities have to be generated. It is investigated whether a problem occurs in the software when an input with this generated cases is made.

The OWASP Top 10 list 2017 [OWA17b] postulated that SQL injections are the vulnerability which occurs in the most applications. Several tools has been developed to detect SQL injections in web applications [DT16]. Huang et al. [HYH+04] developed a tool called WebSSARI which detects vulnerabilities through static analysis methods on the source code and by runtime inspections. There exist other detection tools for example Sania by Kosuga et al. [KKH+07] or the framework WAVES of Hung et al. [HTL+05]. These tools detect the vulnerabilities with different approaches (syntactic, semantic analysis or black-box approach). Every SQL query is parsed into a tree by Sina. For every query a tree has to be generated and stored. Every input query produces a new tree. An attack can be recognized if the new generated tree and the stored tree show differences. The framework WAVES tries to recognize SQL injections and Cross-Site-Scripting (XSS) with the black-box approach. This tool detects vulnerabilities in applications and includes an error scanner.

Spring Boot provides its own security framework which can add authentication and authorization in web applications [Siv17]. These two principles are primary aspects of increasing the security of web applications [Siv17].

In contrast to these papers, the focus of this thesis is not centered on the detection of vulnerabilities which frequently occur on websites (e.g., SQL injections or XSS) according to OWASP. Because the CD pipeline tools are from third-party providers, the thesis only considers the following aspects on the CD pipeline tools: security configurations, access rights, visible sensitive data on the user interface (UI) and connection to other components in the CD pipeline.

## 3.2. Securing pipelines

"A key challenge in a continuous deployment pipeline is the security of the pipeline itself" [RZB+15]. Several persons have developed approaches for securing the pipelines. Bass et al. [BHR+15], Ullah et al. [URS+17], and Rimba et al. [RZB+15] developed approaches in form of tactics to increase the security level of continuous deployment pipelines.

Bass et al. [BHR+15] mapped out an engineering process for hardening the security of a pipeline. In their case, four steps have to be undertaken to increase the security level of the pipeline. In the first step, the collection of the security requirements has to be done. The next step is to "identify the trustworthy and untrustworthy components of the pipeline" [BHR+15]. Bass et al. mentioned that the detection is complex because of the variety of tools which are used in a pipeline. Every tool has its own vulnerabilities.

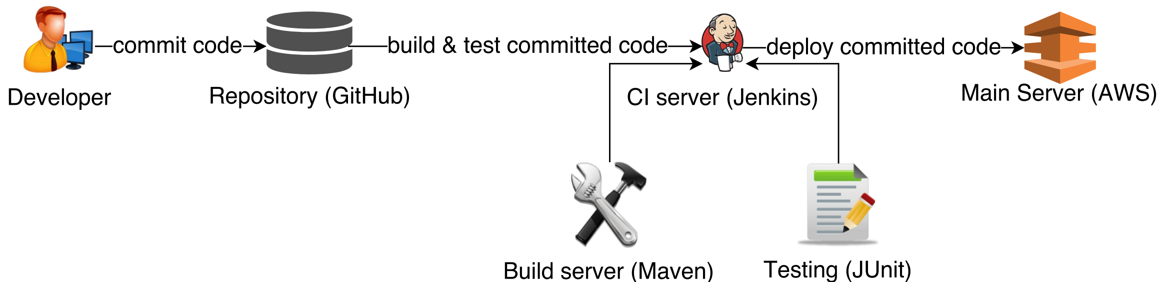



**Figure 3.1.:** Continuous deployment pipeline used in the paper of Ullah et al. [URS+17]

In this work, the investigated pipeline tools are Chef, Jenkins, Docker, GitHub, and AWS. The focus in the work of Bass et al. is only placed on Jenkins. The result of their work is that they give untrustworthy components lower rights. Through the limitation of access and permissions they try to make the pipeline secure. As a result of this, the attacker can only access the trusted components. These components should be able to prevent the attacks.

Rimba et al. [RZB+15] developed four tactics (connect, disconnect, create, and delete). Each tactic refers to either the components or connection of the application. All tactics refer to the connections of a system. The composition of these tactics builds a design of a secure system. The composition is called designed fragment. This can ensure that the security of continuous deployment pipelines are guaranteed. The iterative addition of those tactics increases the security and an attacker can be defended in those aspects. They conducted a case study with a real-time system to prove and verify their tactics. They found out that the combination of security patterns can help to secure the pipeline. The investigated real-time pipeline includes Jenkins and its components are stored on AWS instances [RZB+15].

To secure a continuous deployment pipeline Ullah et al. [URS+17] were developing five security tactics. If the tactics are followed in the handling of continuous deployment pipelines, it prevents an attacker from injecting malicious code in the continuous deployment pipeline. In this paper, two continuous deployment pipelines were compared. The first tactic is implemented in both continuous deployment pipelines. The rest of the tactics are only included in the second continuous deployment pipeline which represents the secure one. The five tactics are: “1. Securing repository through controlled access [...]. 2. Securing connection to the main server through use of private key over Secure Shell (SSH). 3. Using roles on the main server to control access [...]. 4. Setting up the CI server to start up a Virtual Machine (VM) with a clean state [...]. 5. Using Jenkins roles plug-in [...]” [URS+17]. The structure of the continuous deployment pipelines is illustrated in Figure 3.1. The continuous deployment pipelines consist of the following components: GitHub, Jenkins, JUnit and Maven [URS+17]. All components

are hosted on an AWS server except the GitHub repository. Ullah et al. gave no reason why they chose this special structure of continuous deployment pipeline. The considered security aspects of this work are access control and visualization. The evaluation of their implemented tactics were done with two analysis methods. The first one is a qualitative analysis of continuous deployment pipeline. With the goal structuring notation (GSN) they found out that their tactics improve the security level of the continuous deployment pipeline. The quantitative analysis which was conducted with the tools OWASP Scan and OWASP Zed Attack Proxy Scanner provided the results which demonstrated that the secure continuous deployment pipeline has fewer vulnerabilities than the non-secure pipeline. The results of the work also show that the GitHub repository and CI Server (Jenkins) without these implemented tactics have more vulnerabilities than the main server (AWS) [URS+17].

Gruhn et al. [GHJ13] investigated public CI services. Their aim is to detect possible attack vectors with security analysis methods in these services. In a proof of concept they mention that they want to eliminate one class of attack vectors with visualization.

In contrast to these papers, the aim of this thesis is not to find tactics to secure the CD pipelines. The aim is to identify vulnerabilities that violate the security attributes in CD pipelines. These vulnerabilities are discovered with selected tools, among others. The investigated CD pipelines are used in an industrial context.

## 3.3. Threat modeling

According to Möckel and Abdallah [MA10], the threat modeling approach is one essential approach to add security to the SDL. They mentioned that tools such as Microsoft SDL help to understand the threat modeling process and to visualize the threats. Möckel and Abdallah found out in a case study with an online banking system that threat modeling tools are useful but there is an area of freedom in interpreting the results. By using these tools the results are reproducible. Additionally, they discovered that the choice of tools depends on the environment and the user.

Besides, there exists the paper of Abomhara et al. [AGK15] which investigated the threats in a telehealth system. They used the STRIDE method to detect threats at the component level of the system. The approach helps to ensure system security. Abomhara et al. found out that the STRIDE method simplified the detection of vulnerabilities in this telehealth system.

In addition to that mentioned paper, there exists the work of Lipke [Lip17] which is about securing the software supply chain. The thesis shows how to build a secure software supply chain using Docker. Lipke was occupied with the detection of threats in Docker components (e.g., Docker image, Docker registry). With the threat modeling approach method STRIDE he detected threats in those components. He discovered

that the software supply chain cannot be completely secured because not all threats which are detected can be mitigated or eliminated. His evaluation indicated that the critical components of his investigated pipeline are the build server and the production environment. The thesis of Lipke only detected the vulnerabilities with the threat modeling approach method STRIDE.

In contrast to Lipke, the aim of this thesis is also to search for tools that can detect vulnerabilities in the CD pipeline structure. The mentioned papers show that the threat modeling approach is used to detect threats and vulnerabilities in critical systems with the aim to ensure the system security. For this reason, in this thesis the threat modeling approach is not being evaluated but the approach is actively employed to detect the threats and vulnerabilities in CD pipelines.

## 3.4. DevSecOps

In Cater's interview [Car17] Francois Raynaud mentioned that the use of security tools during the deployment process is necessary to add security to the software.

Rahman and Williams [UW16] found out that the automation of activities such as monitoring, testing, and code review add security practices to the DevOps process and can have a positive impact on the security of the system. A further result of their research is that the choice of deployment tools and software metrics have an impact on the security of the system. In addition, Rahman and Williams demonstrated that in eight evaluated companies many DevOps security activities are performed in a non-automated manner. Security requirements analysis, performing security policies or input validation are three activities of that list.

Jim Bird [Bir16] described in his book different possibilities of how to add security practices to DevOps tools and to the development process. He mentioned that security tests and practices can be added to each stage of the pipeline. He recommended that before the source code is checked in threat modeling or peer code review should be performed. SAST tools should be executed in the commit stage. In the acceptance stage tools such as Puppet, Chef or Docker should be used to automate the configuration management. This leads to more security in the process. In this stage, security tests should be performed such as fuzzing or DAST tests. Furthermore, automated security attacks (penetration testing) can be performed to detect further vulnerabilities. In the production stage, he recommended doing monitoring and automated configuration management to detect vulnerabilities. These practices and test methods are primarily used to secure the source code and the development process. The aim is to program securely and find out the vulnerabilities as early as possible. In addition, Jim Bird mentioned methods for securing the software supply chain. A huge amount of applications are open source or third-party components. Therefore, it is necessary to detect the dependencies

between the used applications. The problem is that everyday vulnerabilities are reported in open source software. If third-party components are used, then the application is dependent on these components. If such components have vulnerabilities, then this application has them too [Bir16]. If a company uses Docker, Jim Bird recommended doing dependency checks in Docker images. For securing CD pipelines Jim Bird mentioned that it is necessary to review files and manifests, like Puppet and Docker files. Additionally, he pointed out that looking into the source code or other files is necessary to find secret credentials. In his eyes, one solution is to reduce the attack surface. This means that unused components which have known vulnerabilities should be removed. In his opinion, further aspects such as monitoring of sensitive data, logs and environments (e.g., production, deployment, and testing environment) are necessary. To sum up, Jim Bird's book lists and mentions methods which can be used to detect the vulnerabilities of a CD pipeline. The book gives an overview of tools which can be integrated into the CD pipelines.

In contrary to the book, this thesis detects the vulnerabilities with the threat modeling approach and considers how existing tools can help to detect these vulnerabilities. Jim Bird does not list all tools which can be added to the CD pipeline. The book provides the basis which tools can be integrated into a CD pipeline. The focus of this work is not on identifying vulnerabilities in the developed source code or in the entire agile development process.

Schneider [Sch15] and Storms [Sto15] presented tools and methods which can be used to detect vulnerabilities in applications and which can be used to increase the security level in the DevOps approach. These two presentations are one basis for the Chapter 6.

Another paper of Shu [SGE17] investigated the Docker containers (images) which are present in the Docker Hub registry. They checked more than 300,000 images for vulnerabilities with the security tool Clair[1]. The result of this paper is that on the average the tool Claire detects 180 vulnerabilities in each image.

Kuusela [Kuu17] found out in his thesis how to integrate available software security tools into a CI process. Based on the literature and the documentation of the tools, he identified characteristics to decide whether the tools are suitable for the CI process. The tools he has found are limited to open source/free software which should be easy to integrate and the tools results should be understandable. He has done four case studies in which different tools have been tested. The decision which tools should be integrated is made by the team members of the investigated projects. The tested software were web applications. The following tools were evaluated: Brakeman, FindSecurityBugs, OWASP dependency checks, Version Maven Plugin and Retrie.js. In the case studies, only static analysis tools and dependency checks were integrated and tested. Kuusela discovered

[1] https://github.com/coreos/clair

that these two methods can be easily integrated into the CI process. In addition, the vulnerabilities identified by the tools can be easily eliminated or mitigated.

In contrast to that, this thesis focuses on the integration of tools which detect the vulnerabilities in the essential elements and structure of CD pipelines. Additionally, there is no consideration of vulnerabilities in the developed source code. Static analysis tools which inspect the developed source code are not of interest.

Besides to the mentioned papers, Stažić [Sta17] addressed the addition of security to the agile development process.

In this thesis, the focus does not rest on the entire agile automated development process, but rather on the CD pipeline itself, which is a part of it.

Chapter 4

# Survey on the knowledge of developers

The investigated CD pipelines (see Chapter 7) are pipelines of an industrial software company. To detect vulnerabilities in these CD pipelines, it is necessary to know what developers think and know about this thesis topic. In order to get the knowledge of the company's developers, a survey was conducted in form of an in-depth online interview. In the first part of the following section, it is described what the qualitative research method is and why this method is chosen to gain the knowledge of the developers (Section 4.1). In addition, the chapter includes the description and presentation of the design (Section 4.2), execution (Section 4.3), and results of the survey (Section 4.4).

## 4.1. Qualitative research method

If a topic is not known then a qualitative approach can help to obtain basic knowledge about it [MW05]. Mack et al. [MW05] recommended the qualitative research method to understand the present problem or the context of a topic. The method obtains the experiences and opinions of a sample of the population. In addition, Mack et al. mentioned that the gained data is used to describe the topic or problem and not to predict or quantify the data. They discovered that the most common method to do qualitative research is the in-depth interview. An interview consists mainly of open questions, this means that the participants answer them with their own words and not only with yes and no. It is important to formulate the questions in such a way that the participants cannot answer with yes or no. According to Mack et al., the researchers gain an overview of this method and deeper insight into the topic and through the open-ended questions it does not restrict the participant way of thinking.

At the beginning of the research of this thesis topic, the in-depth interview method should help to obtain the knowledge of developers of a software company. Due to the

fact that the employees have little time to spare, the interview is designed in form of an online survey.

## 4.2. Survey design and questions

To gain the knowledge of the employees two different surveys were designed. With the first survey the knowledge of a sample of employees is obtained. The second survey is structured as follows. The first part of this survey contains the same questions as the first survey. The second part includes a specific question about the CD pipelines which are examined in the case study. The second survey is only sent to the employees who are team members of the projects using the investigated CD pipelines of the case study. The Appendix A includes the complete questionnaires of both surveys.

Nine questions are the same in both surveys. The first four questions are about various aspects of CD pipelines (security goals, security attributes, attack scenarios). The next five questions are used to gain the profiles of the participants. The first four questions are:

1. In your opinion which security objectives should be pursued to CD pipelines? Please do not focus on a specific used pipeline. Think in general.
2. In your opinion which security attribute is the most important one in respect to CD pipelines (artifacts, files, scripts, connections, ...)? Order the following security attributes (confidentiality, integrity, availability, authorization, authentication, nonrepudiation) according to their importance. The attribute on top is the most important one for you.
3. In your opinion what are possible attack scenarios for the pipeline you use? Against which attacks would you like to protect your pipeline?
4. Which security objectives are pursued in your project in respect to CD pipelines? Which are implemented?

The first question should give an overview about the employee's thinking in regard to the security of CD pipelines. The gained data of the second question should help to delimit the subject because it reflects the interest and the thought necessity of the employees. The questions in the second part are mostly multiple-choice questions. These questions provide the researcher with information about the participants themselves. The questions are the following:

5. How many years of experience in software development do you approximately have?

6. Which tools do you know and/or use?
   Response options: (DevOps tools) Jenkins; Kubernetes; TeamCity; Spinnaker; Travis; GoCD; Concourse CI; JFrog Artifactory; (static analysis tools) PMD; Checkstyle; FindBugs; FindBugs Security; (security tools) OWASP ZAP; BDD Security; JFrog Xray; Security Monkey; Black Duck; Snyk
7. In which role do you interact with your CD pipeline?
   Response options: user (committing code to the project, usage of the CD pipeline); installation and operation of the pipeline; configuration of the pipeline; other
8. In your opinion how important is the topic security vulnerabilities in CD pipelines?
   Response options: 1; 2; 3; 4; 5 (1: not important, 5: very important)
9. How often do you deal with security in your development process?
   Response options: Never; only occasionally; quite often; most of the time; no answer

The fifth, seventh and ninth questions are asked to find out how familiar employees are with security and how often they come into contact with the CD pipeline. The sixth question shows to the researcher how much knowledge the employees have in different DevSecOps tool categories. Since there is a huge number of DevSecOps tools [Xeb], the selection of tools is made for those which are used and known by individual employees of the company or that are known to the researcher up to this point in time. The static analysis tools are asked as well because the company uses these tools in almost every project to detect errors in the source code. Question eight shows to the researcher what priority the security has in their thinking. The additional question of the second part is the following:

10. In the next step think about the security of the [...] CD pipeline. In your opinion how secure is this pipeline?
    Response opinion: 1; 2; 3; 4; 5 (1: means CD pipeline is insecure, 5: means CD pipeline is secure (pipeline has no vulnerabilities))

The results of these questions create the base for Chapter 5.

## 4.3. Survey execution

The online surveys were sent by e-mail to approximately 100 employees of a selected industrial company which includes the investigated CD pipeline project teams of the case study. The participation is on a voluntary basis and can be performed at any time. 59 employees participated in the survey only partially. 19 persons answered all questions. Some persons mentioned that they have no idea how to answer the free text questions

because they are not familiar with the topic of this thesis or have no background in this context. In the further course and evaluation, only the fully 19 answered surveys are used.

## 4.4. Survey results

In the next part the results of the survey are visualized and described. First, the results of the questions from the second part of the survey are presented (question number 5 to 9). These results will provide an overview of the employees existing expertise. After that, the results of the questions from the first part of the survey are shown (question number 1 to 4) which presents the knowledge on attack surfaces and security objectives in industrial project pipelines. The results of these four questions create the basis for Chapter 5. The results of question 10 are presented in the case study chapter (see Chapter 7).

### 4.4.1. The results of the questions from the second part of the survey

The **results of survey question 5** show an answer to how many years of software development experience the participants have. The results show that on average the participants have 10.32 years of development experience. The complete distribution is included in the appendix (see Figure A.2).

The **results of survey question 6**, which are presented in Figure 4.1, show an answer to which tools are known or used by the participants. It can be seen that the CI server Jenkins is known by nearly all participants. Other CI tools such as GoCD or Concourse CI are not that well-known among these industrial employees. The containerization tool Kubernetes is known by half of the participants. One third of the participants approximately know the artifact repository JFrog. The deployment tool Spinnaker is not known to any participant. The radar chart clearly shows that the know-how in security testing tools (OWASP Zap, BBD Security, JFrog Xray, Security Monkey, Black Duck, and Snyk) is not as well presented as in static analysis tools (Checksystle, PMD, and FindBugs). In summary, it can be said that the competence of the industrial company lies in the CI tool Jenkins and in the static analysis tools. There exists not much know-how in security tools which can be used to secure applications and components of CD pipelines.

The **results of survey question 7** answer the question in which role the participants interact with the CD pipeline. The results are presented in Figure 4.2 and show that 58% of the 19 participants interact with all facets of a pipeline. Firstly, these 58% of the participants use the pipeline. Secondly, they have the rights to install, configure, and operate the pipeline. 16% of the participants use and configure the pipeline. In addition,

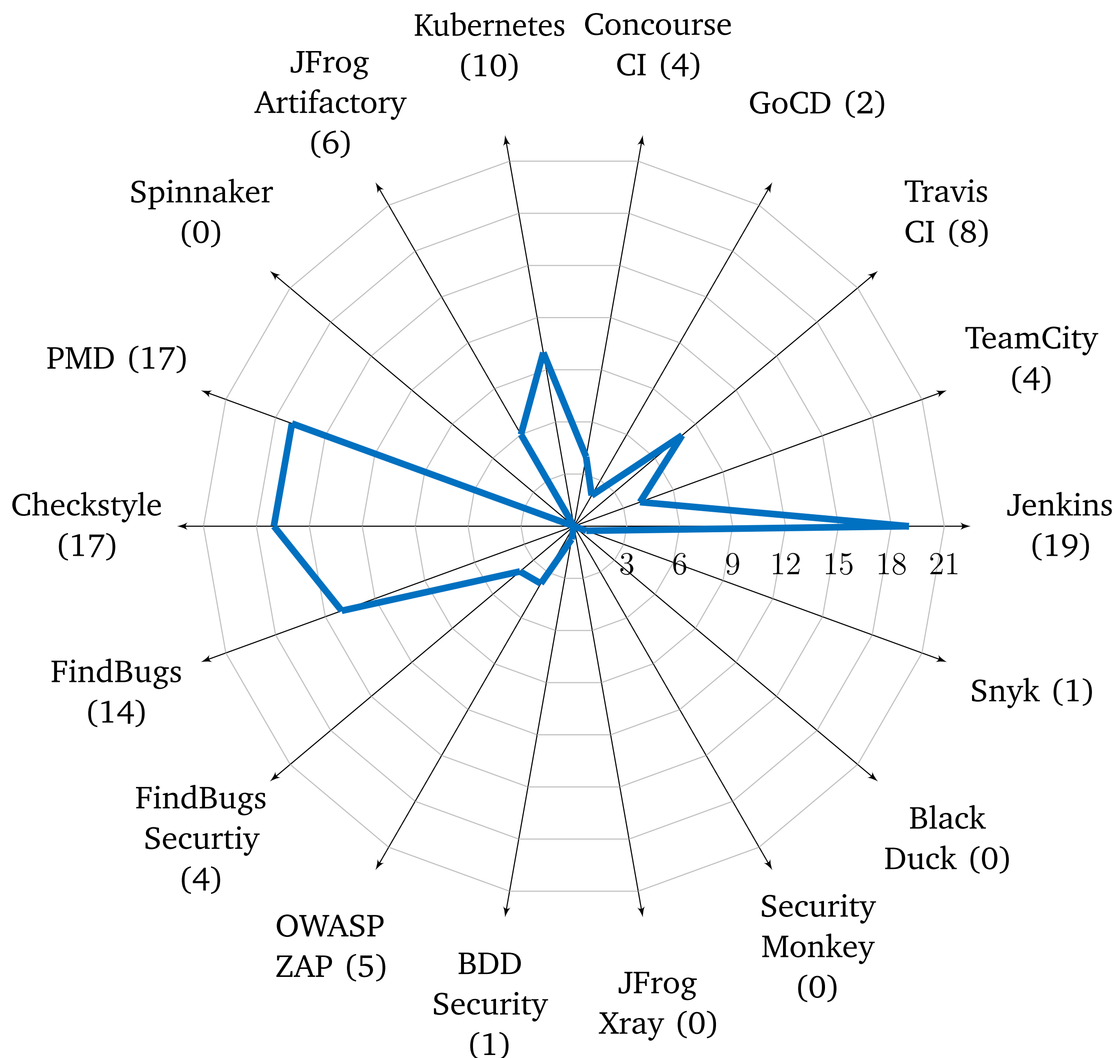



**Figure 4.1.:** Survey question 6 results: Tools which are known and/or used

one person (5%) is able to install, operate the pipeline, and use it. Further 16% of the participants only use the pipeline. This means that they can commit code changes and have access to the UI of the CD pipeline components. Only one person does not come into contact with the pipeline because he is scrum master and manages the team. In summary, it can be said that 99% of the participants come into frequent contact with the pipeline.

The **results of survey question 8** show an answer to the question as how important CD pipeline security is seen by the participants. The results are shown in Figure 4.3. The figure represents that the employees realize that it is necessary to do research on this topic. 42% of the participants think that the consideration of this topic is fairly important (4). 16% think that the observation is very important (5). Approximately

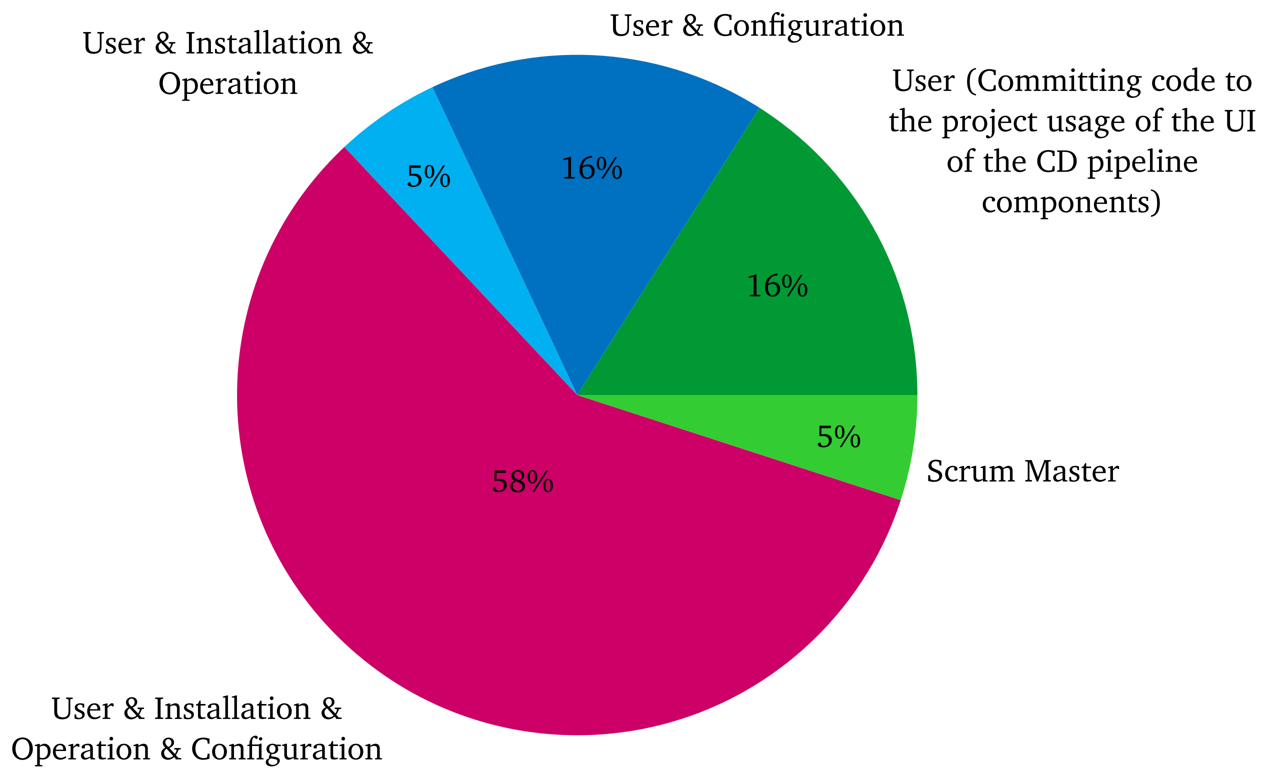



**Figure 4.2.:** Survey question 7 results: The role of the developers if they interact with a CD pipeline.

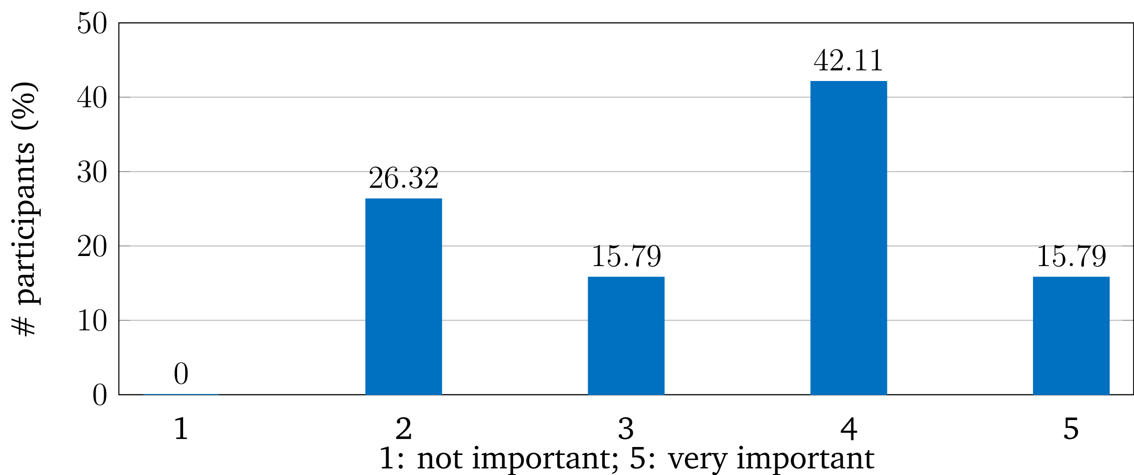



**Figure 4.3.:** Survey question 8 results: The importance of the thesis topic in an industrial company

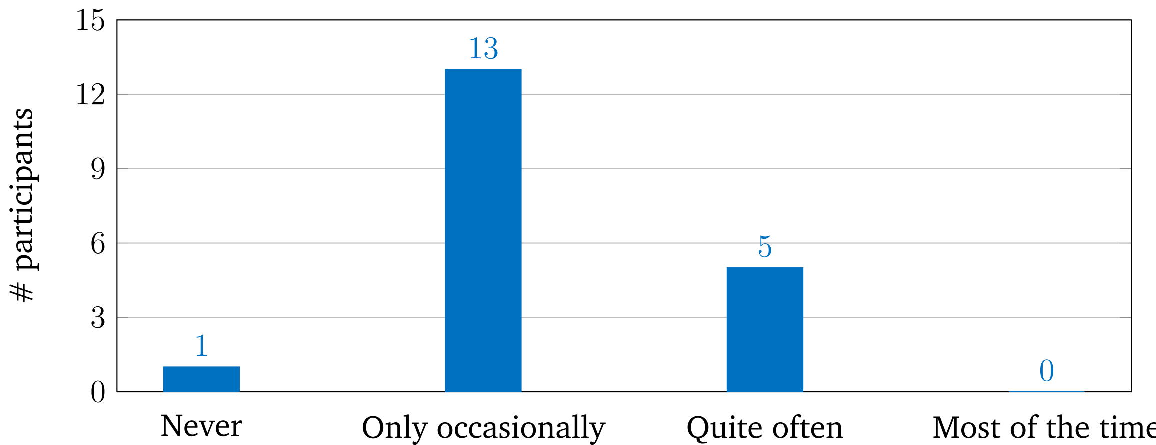



**Figure 4.4.:** Survey question 9 results: The developer's security involvement frequency during the development process

26% of the participants find the topic only slightly important (2). 15% find the topic important (3). To sum up, it can be said that on average the employees find that the topic has a significance of 3.5. This is the mean value between important and fairly important. In the view of the employees it is necessary to deal with the topic.

The **results of survey question 9** show an answer to the question of how often the participants deal with security in the development process. The results are presented in Figure 4.4. The figure shows that the employees only occasionally deal with security topics during their development process. Five employees deal with the security context quite often. One participant mentioned that he is never concerned with security topics. There was no person among the participants who is occupied with security most of the time. In conclusion, it can be said that the employees have not much know-how in this context.

### 4.4.2. The results of the questions from the first part of the survey

The following section will present an extract of the answers of the questions from the first part of the survey.

The **results of survey question 1** describes the attitudes of the employees towards security objectives in CD pipelines in general. All given answers were analyzed and equal answers were aggregated under one heading. Below is a list of the essential and grouped points of the answers to the question.

- No pipeline modification through users who have no access rights
- No triggering of the pipeline through unauthorized persons

- Securing source code, logs and artifacts
- Securing environment properties such as login data
- Securing credentials (encrypt all sensitive data)
- Build steps should not be manipulated
- No vulnerabilities in dependencies
- Reduce human errors (storing password)
- Secure transmission over Hypertext Transfer Protocol Secure (HTTPS) or Secure Shell (SSH)
- Use 4-eye-principle
- Check access rights of the components of the CD pipeline

The **results of survey question 2** are presented in Figure 4.5. This figure shows what security attributes employees of an industrial company consider to be the most important. To get these results the participants had to rank the six security attributes. The importance is calculated out of the ranking and by the means of the following formula:

$$\begin{aligned} ImportanceOfAttribute = {} & valueOfRank1 * 5 + valueOfRank2 * 4 + \\ & valueOfRank3 * 3 + valueOfRank4 * 2 + \\ & valueOfRank5 * 1 + valueOfRank6 * 0 \end{aligned} \tag{4.1}$$

The *valueOfRank* 1 to 6 are the results of the ranking of the survey. The individual values are listed in a matrix (see Figure A.1). The resulting ranking shows that the attributes which belong to persons such as authorization, authentication and nonrepudiation are less important than the CIA triad attributes. It seems, that all people who are interacting with the pipeline considers out of all attributes which belongs to persons only the authentication as important. Out of the six ranked attributes the top three are extracted. In the view of the employees the 3 top ranked are attributes are integrity, availability, and confidentiality. Based on the results of the survey, only the CIA attributes are considered in the further course of the thesis.

The **results of survey question 3** list the attack scenarios which the employees considerer most likely in CD pipelines. The results of them are aggregated, categorized and mapped to the three security attributes integrity, confidentiality, and availability. The contents of these three categories are as presented below.

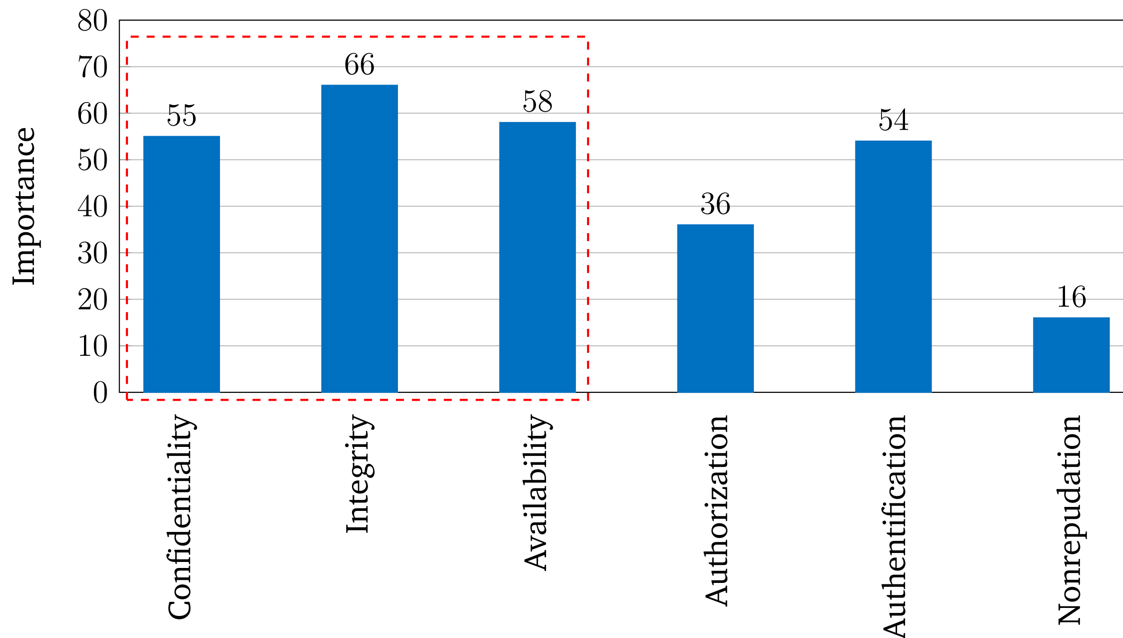



**Figure 4.5.:** Survey question 2 results: Assessment of the six security attributes through developers of a selected company

**Integrity**

- An attacker or a third person who has unauthorized access manipulates the configuration of the pipeline. The manipulation can affect every specific pipeline file like the Jenkinsfile, Dockerfile or on any other configuration like the CD server configuration or any component of the server.
- Manipulation of artifacts, logs or deployment scripts.
- Injection of malicious code, files which can include worms or viruses into the CD pipeline. These files can be injected through back doors or leaks in the application. It is possible that such malicious code is deployed.
- In many cases, the used pipeline tools have vulnerabilities and open new doors for potential attackers.

**Availability**

- DoS attacks - effectively shut down the server.
- An unavailable pipeline would prevent the delivery of the software.

- Attacks which manage to change something on the pipeline can damage the environment in which the pipeline is running.

**Confidentiality**

- Execute a MITM attack.
- Cross build injection attack.
- An attacker can gain sensitive data such as credentials if used plugins, libraries or pipeline components have vulnerabilities.

These attack scenarios establish a basis to detect the threats and vulnerabilities in CD pipelines.

The **results of survey question 4** point out which security objectives are pursued in the industrial projects with respect to CD pipelines. The security objectives are aggregated and presented in the following section.

**Security objectives in industrial projects**

- Requiring authentication and authorization
- Securing credentials and hide critical data.
- Review the process
- No information should be included in the source code of applications
- Implemented access control (not all team members have administrator rights)
- Keep the pipeline components and software up to date

It can be seen that in the industrial projects authentication and authorization approaches are implemented. In addition, securing sensitive data and access rights also contributes to secure the pipeline. Two participants mentioned that they have too less or none security objectives. If a project does not pursue security goals, it cannot be guaranteed to the customer that the software will be deployed securely. The results of these four questions help to find the vulnerabilities in CD pipelines and in the further course to investigate the pipelines in the case study. A non-observance of these mentioned security objectives (results of survey question 4) leads to vulnerabilities and open attack entry points into the CD pipeline. In the case study, it is necessary to check which kinds of the security objectives are kept in the investigated CD pipelines.

Chapter 5

# Vulnerabilities in CD pipelines

---

Before vulnerabilities in CD pipelines can be detected, two prerequisites have to be defined. These prerequisites, a generalized pipeline and possible types of threat categories, are described in the first part of this section (Section 5.1). After this, the STRIDE method is then applied to the generalized pipeline to detect threats and vulnerabilities (Section 5.2). The only component which is used in both investigated CD pipelines in the case study (Chapter 7) is Jenkins. Therefore, the following section of this chapter describes the vulnerabilities in Jenkins which can affect CD pipelines' security (Section 5.3). The last section (Section 5.4) sums up the results of the survey (Section 4.4.2) and of this chapter.

## 5.1. Prerequisites for vulnerability detection

The first step in the threat modeling approach is the decomposition of the pipeline in its components and data flows. For the detection of vulnerabilities in CD pipelines, it is necessary to know which components and data flows are present. To detect vulnerabilities, a generalized pipeline is needed. Therefore, in the first part of this section, such a generalized CD pipeline is described.

Vulnerabilities can be derived from threats. In order to detect vulnerabilities, it is necessary to find out who has an interest in attacking the pipeline. The next step is to identify the threats of the CD pipeline. Furthermore, a distinction is made between external and internal threats which are described in the second part.

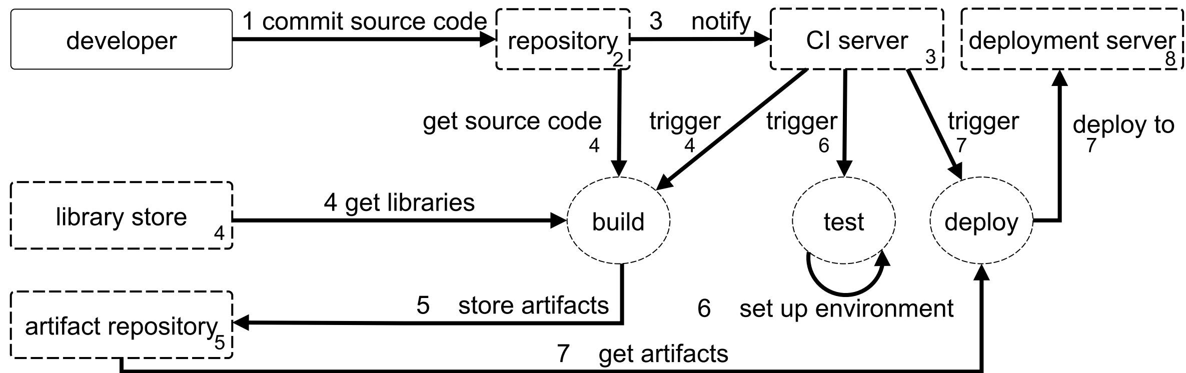



**Figure 5.1.:** Components and data flows of a generalized CD pipeline

### Generalized CD pipeline

Figure 5.1 illustrates the key components and data flows of a generalized CD pipeline. This pipeline is a generalized mixed version of the named pipeline at the beginning (Figure 2.1) and the investigated CD pipelines in Chapter 7.

The components of the pipeline in Figure 5.1 are the source code repository (see 2), the CI server (see 3), the library store (see 4), the artifact repository (see 5) and the deployment server (see 8). The pipeline's processes are as follows (see Figure 5.1): First, a developer commits code changes into the repository (see 1). After that, the CI Server is notified about this changes (see 3). As a result of this, the build step is triggered and the used external libraries are downloaded from the library store (see 4). After that, the code is built. The next step is that the built artifact is stored in the artifact repository (see 5). The sixth step is that the tests are triggered and the environment for the test session is set up (see 6). If the tests are successfully executed and no errors have occurred the deployment step is triggered (see 7). In this step, the artifact is downloaded from the artifact repository and is deployed on a deployment server. If the artifact is available on the deployment server then the customer can install the artifact and use it.

### External vs. insider threats

External threats are located outside of an organization. Criminal organizations, unauthorized persons or foreign adversaries can be a threat for the application [Col17]. These attackers try to find the easiest way to exploit a vulnerability of the application. According to Cole [Col17], the easiest way to launch an attack is if the attacker can acquire the access rights of an insider employee.

In 2017 the SANS Institute [Col17] published the results of an insider threat survey. This survey was intended to provide more insight into insider threats. An insider threat

can be every internal employee or third-party expert. The number of participating organizations is not mentioned, but it is said that a “wide range of organizations” participated [Col17].

The results of the survey point out that 40% of the participating organizations think that malicious insider employees are the most damaging threat vector. Malicious threats can be caused by an employee who wants to harm the company. One possibility to do this is the publication of sensitive customer data. In addition to that, the institute mentions that the companies have to deal with malicious and accidental insider threats. An accidental insider is an employee “who is tricked or manipulated into causing harm or whose credentials have been stolen” [Col17]. His unintended actions is a threat to the company. A malicious insider is an employee who intends to harm a company through an attack.

To sum up, an employee or third-party expert can harm the company in the easiest way if he has negative intentions or he is unaware of what he does through his actions. Cole [Col14] said that external threats occur more often than insider threats. Moreover, he mentioned that it is harder to detect insider threats than external threats.

## 5.2. Threat and vulnerability detection with STRIDE

To detect vulnerabilities in CD pipelines the aforementioned threat modeling approach is used Section 2.5.6. This approach can be detailed detect threats, vulnerabilities, and risks. The ensuing section contains the results of the execution of the threat modeling approach method STRIDE based on the generalized CD pipeline. The results of the survey question 2 show that the top three necessary security attributes in this considered industrial context are: integrity, confidentiality and availability. Consequently, only the threats in the categories tampering (T), information disclosure (I) and denial of service (D) of the STRIDE method are investigated for each component and data flow of this CD pipeline.

In order to consider all possible threats, the cards of the Elevation of Privilege Threat Modeling Card Game [Sho14a], which are issued by Microsoft, are used as a basis. These cards are an easy way to understand the threat modeling approach. In addition, these cards are suitable for those persons who have never dealt with this approach before. Each card contains a part of the STRIDE method and explains a case which can occur. For every STRIDE part several cards are available.

In the following section, each component and data flow is analyzed for threats and vulnerabilities. The arisen effects which the threats would have if they were successful, are also presented. The work of [Sch17] and [BHR+15] confirm some detected vulnerabilities. In a further step, vulnerabilities are derived from the detected threats. According to Shostack [Sho14b], the results of the STRIDE method are presented

in tables. If a pipeline component or stage has vulnerabilities then it is possible for an attacker to exploit them. As reported by Kim et al. [KDWH16], it is possible to scan the network by running injected malicious unit tests. The mentioned attacks in Section 2.6.4 show that it is possible that components of the pipeline can be compromised (e.g., Jenkins server). As a result of this, it is possible that code changes are not delivered through the CD pipeline. All mentioned exploitations of attacks could dissatisfy the customer and most likely damage the image of the company as well.

At first, the detection of the threats and vulnerabilities in the commit stage is performed. Table 5.1, Table 5.2, and Table 5.3 show the results of the execution of the STRIDE method in the commit stage.

Table 5.1 shows which tampering is possible in the commit stage. If there are none or few access restrictions then the attacker can commit arbitrary code or manipulate the pipeline file script (e.g., the Jenkinsfile). He could remove this file or commit an empty version of it. Furthermore, such actions can be executed if no review of commits is practiced. Another vulnerability is when the pipeline script is not tested. The results of this action is that code can include malicious parts and if an empty or not executable pipeline script is committed then there is no delivery of code changes possible.

| Occurrence | Commit stage |
|---|---|
| Threat type | S **T** R I D E |
| Threat | Commit arbitrary code, manipulate or remove pipeline file scripts |
| Effect | Malicious code; no delivery |
| Vulnerability | •None or few access restrictions<br>•No review of source code changes<br>•No testing of pipeline script |

**Table 5.1.:** STRIDE analysis: Tampering in the commit stage

Table 5.2 shows the threats, effects, and vulnerabilities which can occur if in the CD pipeline commit stage information are disclosed. If the attacker would be able to read sensitive data in the commit stage then he could log in to the CD pipeline components. This would be the easiest way to run an attack, because the attacker gains the same privileges as the developers and could perform the same actions as them. Information is disclosed in the commit stage if there would exist sensitive data (e.g., passwords of CD pipeline components) in the commit message or the source code. A unencrypted connection (e.g., HTTP) to the source code repository can also lead to a disclosure of information.

| Occurrence | Commit stage |
|---|---|
| Threat type | S T R **I** D E |
| Threat | Read sensitive data |
| Effect | Attacker gain sensitive data<br>→ same rights as developer |
| Vulnerability | •Commit message or source code includes sensitive data<br>•Unencrypted connection (e.g., HTTP) |

**Table 5.2.:** STRIDE analysis: Information disclosure in the commit stage

Table 5.3 shows the results of the analysis of DoS attacks in the commit stage. No review of code changes or no testing of the pipeline script could allow an attacker to commit an empty or damaged pipeline file script (e.g., the Jenkinsfile). This action can also be executed if the attacker has the same access to the server as the developer. An attacker can gain the same rights if there exist none or few access restrictions. The result of this would be that the pipeline stage cannot be executed.

| Occurrence | Commit stage |
|---|---|
| Threat type | S T R I **D** E |
| Threat | Commit empty pipeline file scripts |
| Effect | Pipeline stages cannot be executed |
| Vulnerability | •None or few access restrictions<br>•No review of source code changes<br>•No testing of pipeline scripts |

**Table 5.3.:** STRIDE analysis: Denial of service attack on the commit stage

Table 5.4, Table 5.5, and Table 5.6 show the results of the vulnerability detection of the component repository.

Table 5.4 shows the threats, effects, and vulnerabilities which can occur if the CD pipeline component repository is tampered with. Similar to the commit stage in the repository the attacker could exploit the same vulnerabilities. If there exists none or few access restrictions then he could obtain access to the repository and commit arbitrary code or an empty or broken pipeline file script (e.g., the Jenkinsfile). In addition, vulnerabilities in the repository version or in used plugins in the repository could allow the attacker to commit those arbitrary files. No checking of the source code means that no one can immediately detect the malicious code. As a result of this, the source code which is versioned in the repository can include malicious parts and it is possible that code changes cannot be delivered because of damaged pipeline file scripts. Another

threat is that the attacker can change the security configurations of the repository and thereby would open the server to unauthorized individuals. He could do this if he gains administration rights or the repository has none or few access restrictions. If the attacker has admin rights then he also can change the webhook URL. This means that if the developers would commit code changes either no or another server is triggered.

| Occurrence | Repository |
|---|---|
| Threat type | S **T** R I D E |
| Threat | Commit arbitrary, manipulated pipeline file scripts or remove them |
| Effect | Malicious code changes; no delivery |
| Vulnerability | •None or few access restrictions<br>•No review of code changes<br>•Vulnerable version of the repository<br>•Use vulnerable plugins |
| Threat | Change security configurations of the repository |
| Effect | Unauthorized persons get access to repository |
| Vulnerability | •None or few access restrictions |
| Threat | Change webhook URL (e.g., CI server URL) |
| Effect | Trigger another CI server e.g.; no delivery |
| Vulnerability | •None or few access restrictions |

**Table 5.4.:** STRIDE analysis: Tampering in the repository

Table 5.5 shows the threats, effects, and vulnerabilities which can occur if in the repository information are disclosed. If an attacker can gain administrator rights through none or few access restrictions then he can read webhook URLs. The results of this is that the attacker knows the CI server URL and its environment.

| Occurrence | Repository |
|---|---|
| Threat type | S T R **I** D E |
| Threat | Read sensitive data (e.g., CI server URL) |
| Effect | Attackers gain sensitive data |
| Vulnerability | •None or few access restrictions |

**Table 5.5.:** STRIDE analysis: Information disclosure in the repository

Table 5.6 shows the results of the analysis of DoS attacks on the repository. If the repository has none or few access restrictions and the transferred source code is not

scanned, an attacker can modify or remove relevant pipeline scripts (e.g., the Jenkinsfile). As a result it could be possible that there would be no delivery of code changes. If the repository would be installed in a Docker container and the attacker gains access to the environment then it is possible that an attacker shuts down the repository. If the repository is not available then no commit of code changes is possible and the delivery process would be stopped. As an example, there is no software development team work possible. It is necessary to have a secure repository environment and access rights to prevent such actions. Another threat would be a change of the webhook and therefore another or no pipeline can be triggered. This is possible, if the attacker would gain administrator rights or the repository has none or few access restrictions.

| Occurrence | Repository |
|---|---|
| Threat type | S T R I **D** E |
| Threat | Change or remove pipeline file scripts |
| Effect | No delivery of code changes, team work stopped |
| Vulnerability | •None or few access restrictions<br>•No review of source code changes |
| Threat | Shut down repository |
| Effect | No commits are possible, delivery process stopped |
| Vulnerability | •Insecure environment of the server<br>•None or few access restrictions |
| Threat | Change webhook |
| Effect | Trigger another or no pipeline |
| Vulnerability | •None or few access restrictions |

**Table 5.6.:** STRIDE analysis: Denial of service attack on the repository

Table 5.7, Table 5.8, and Table 5.9 show the results of the vulnerability detection in the CI server component.

Table 5.7 shows the threats, effects, and vulnerabilities which can occur if the CD pipeline component CI server is tampered with. If the CI server — for example Jenkins — is installed in a Docker container and an attacker has got access to this environment then it would be possible to change the configuration file of the CI server. How this attack would work is described in Section 5.3.1. The result of this attack would be that the CI server User interface (UI) could be opened up to unauthorized persons. All persons who have the CI server URL can trigger the pipeline or remove pipeline steps or relevant files. If an attacker would gain administrator rights then he can change the credentials or the security configurations of the CI server. This action for the Jenkins UI is shown in Section 5.3.2. As a result, an attacker could prevent the developers from

logging on to the CI server or restrict their access. This action can be executed if the CI server has none or few access restrictions. According to Schirrmacher [Sch17], since October 2017 all Jenkins versions until 2.84 and 2.73.2 are vulnerable. This means that attackers would possibly be able to inject malicious code. In addition to that, Bass et al. [BHR+15] mentioned that artifact and image builders have vulnerabilities. As a result of this an attacker can manipulate the CI server (e.g., pipeline jobs). This means that other repositories and code builds could be triggered and deployed or the Jenkinsfile could be replaced by another Jenkinsfile. It is also possible that through the attack a execution of the pipeline would be prevented. An attacker can remove or display sensitive data (e.g., logs, config files) if he has the access rights to do that.

| Occurrence | CI server |
|---|---|
| Threat type | S **T** R I D E |
| Threat | Change config file |
| Effect | Open CI server UI for unauthorized persons |
| Vulnerability | •Insecure CI server environment (see change Jenkins config file Section 5.3.1) |
| Threat | Change credentials and security configurations |
| Effect | Prohibit other persons from accessing |
| Vulnerability | •None or few access restrictions (see Section 5.3.2) |
| Threat | Manipulate CI server (e.g., pipeline jobs) |
| Effect | Malicious source code, no delivery |
| Vulnerability | •Vulnerable version of the CI server<br>•Use vulnerable plugins (e.g., vulnerabilities in Jenkins version and plugin in October 2017 [Sch17])<br>•Artifact or image builder have vulnerabilities [BHR+15] |
| Threat | Remove or print sensitive data or files (see Section 5.3.3) |
| Effect | Necessary files are deleted; unauthorized persons can gain access to pipeline components |
| Vulnerability | •None or few access restrictions |

**Table 5.7.:** STRIDE analysis: Tampering in the CI server

It is possible to create a parallel pipeline which is triggered for example every 15 minutes and deletes relevant files. How this attack works is described in Section 5.3.3. As a result of this necessary files can be deleted and it is possible that unauthorized individuals can gain access to CD pipeline components because the sensitive data is visible. The vulnerability which lies behind that is that there exist none or few access restrictions.

Table 5.8 shows the threats, effects, and vulnerabilities which can occur if on the CI server information are disclosed. If sensitive data would be shown on the CI server UI then it is possible for an attacker to read this sensitive data (e.g., user or pipeline component credentials). As a result of this the attacker could gain access to the CD pipeline components such as an authorized user. This action is possible if the CI server has none or few access restrictions.

| Occurrence | CI server |
|---|---|
| Threat type | S T R **I** D E |
| Threat | Read sensitive data (e.g., credentials) |
| Effect | Gain access to pipeline components |
| Vulnerability | •None or few access restrictions<br>•Sensitive data is shown on the CI server UI (plain text) |

**Table 5.8.:** STRIDE analysis: Information disclosure in the CI server

Denial of Service attacks which are feasible on the CI server are listed in Table 5.9. If the pipeline jobs would be changed or removed then this means that pipeline file scripts (e.g., the Jenkinsfile) could be removed or changed. This triggers a new pipeline instance and a new or no delivery process would be started. Another threat is that it is possible that the CI server is shut or slow down (see compromised CD pipeline components Section 2.6.4).

| Occurrence | CI server |
|---|---|
| Threat type | S T R I **D** E |
| Threat | Change or remove pipeline jobs |
| Effect | Start no or other delivery process |
| Vulnerability | •None or few access restrictions |
| Threat | Shut or slow down CI server (see Section 2.6.4) |
| Effect | No delivery of code changes; get sensitive data; execute malicious software |
| Vulnerability | •None or few access restrictions<br>•Insecure CI server environment<br>•Vulnerable CI server version |

**Table 5.9.:** STRIDE analysis: Denial of service attack on the CI server

If this action would be performed then code changes cannot be delivered. In addition, an attacker can get sensitive data or execute malicious software on the CD

pipeline component. This effect is possible caused if the component has none or few access restrictions or a vulnerable CI server version is in use.

If the CI server runs in a Docker container and the CI server environment is insecure then it is possible for an attacker to shut down the server. If the application has none or few access restrictions then the attacker could likely shut down the Jenkins server, for example via another URL input.

Table 5.10, Table 5.11, and Table 5.12 show the results of the vulnerability detection of the build stage.

Table 5.10 shows the threats, effects, and vulnerabilities which can occur if the pipeline build stage is tampered with. If the developer has used vulnerable dependencies and the application is built then an attacker can inject arbitrary, malicious source code. A result of this is that the injected source code can be executed during the build process and can harm the CD pipeline. This is possible if an MITM attack is done. In addition to that, another threat occurs if the used libraries can be replaced in the library store or during data transmission. This action is possible if an unencrypted connection to the remote library (e.g., Maven Central) exists. Nowadays a checksum is generated and stored for every library. If the library is downloaded from the remote repository and the checksum is not verified then it is possible to replace the libraries. As a result of this, other malicious libraries can be built and malicious code can be executed.

| Occurrence | Build stage |
|---|---|
| Threat type | S **T** R I D E |
| Threat | Replace libraries during data transmission |
| Effect | During the build process arbitrary libraries can be executed |
| Vulnerability | •Unencrypted connection (e.g., HTTP)<br>•No library verification (e.g., checksum) |

**Table 5.10.:** STRIDE analysis: Tampering in the build stage

Table 5.11 shows the threats, effects, and vulnerabilities which can occur if in the pipeline build stage information are disclosed. If the attacker can read the transferred data during the build process and no secrets are uncovered then the vulnerability of an unencrypted connection (e.g., HTTP) exists but this action would not cause any damage to the CD pipeline itself.

| Occurrence | Build stage |
|---|---|
| Threat type | S T R **I** D E |
| Threat | Read transferred data |
| Effect | If no secrets are uncovered then no damage effects on CD pipelines |
| Vulnerability | •Unencrypted connection (e.g., HTTP) |

**Table 5.11.:** STRIDE analysis: Information disclosure in the build stage

Table 5.12 shows the results of the analysis of DoS attacks on the build stage. If the attacker has the possibility to inject a manipulated library which includes commands to shut down the build server or to remove the pipeline file script then no delivery of code changes would be possible. It is possible that an attacker is able to include a key logger in a library which tracks secrets during the build process. This action is possible if there exists an unencrypted connection or no library checking is done.

| Occurrence | Build stage |
|---|---|
| Threat type | S T R I **D** E |
| Threat | Manipulated library includes command to shut down server or to remove the pipeline file script |
| Effect | No delivery of code changes; track secrets |
| Vulnerability | •Unencrypted connection (e.g., HTTP)<br>•No library checking (e.g., checksum) |

**Table 5.12.:** STRIDE analysis: Denial of service attack on the build stage

Table 5.13, Table 5.14, and Table 5.15 show the results of the vulnerability detection of the component artifact repository.

Table 5.13 shows the threats, effects, and vulnerabilities which can occur if the pipeline component artifact repository is tampered with. If an attacker could upload vulnerable artifacts into the artifact repository then this action is possible because there are none or few access restrictions. Thus damaged artifacts can be deployed. A further threat is that an attacker can manipulate the artifact repository. Assuming that there exists a vulnerable artifact repository or vulnerable plugins are used then an attacker can exploit the vulnerabilities it contains. If there exists none or few access restrictions then an attacker can change the security configurations and open the repository for unauthorized persons who know the URL of this server.

| Occurrence | Artifact repository |
|---|---|
| Threat type | S **T** R I D E |
| Threat | Upload vulnerable artifacts |
| Effect | Deploy vulnerable artifacts |
| Vulnerability | •None or few access restrictions<br>•Vulnerable artifacts (no testing of artifacts) |
| Threat | Attacker can manipulate artifact repository |
| Effect | Deploy malicious artifacts or no artifacts are available |
| Vulnerability | •Vulnerable version of the artifact repository<br>•Use vulnerable plugins |
| Threat | Change security configurations |
| Effect | Make artifact repository accessible to unauthorized persons |
| Vulnerability | •None or few access restrictions |

**Table 5.13.:** STRIDE analysis: Tampering in the artifact repository

Table 5.14 shows the threats, effects, and vulnerabilities which can occur if in the artifact repository information are disclosed. If the connection is unencrypted (e.g., HTTP) then an attacker can read the artifact name but it would not cause any damage to the CD pipelines.

| Occurrence | Artifact repository |
|---|---|
| Threat type | S T R **I** D E |
| Threat | Read artifacts |
| Effect | No damage effects on CD pipelines |
| Vulnerability | •Unencrypted connection (e.g., HTTP) |

**Table 5.14.:** STRIDE analysis: Information disclosure in the artifact repository

If the artifact repository is hit with a DoS attack then the server could be crashed and no delivery of artifacts would be possible any more. This analysis is shown in Table 5.15. From this knowledge two threats and three vulnerabilities can be derived. The threats are that the server can be shut down or an attacker can upload a huge amount of artifacts which slow down the server. This is possible if there exist none or few access restrictions or the environment where the server is installed is insecure. If there is no check of the uploaded data, the attacker has the freedom to upload the desired amount of data.

| Occurrence | Artifact repository |
|---|---|
| Threat type | S T R I **D** E |
| Threat | Shut down artifact repository server |
| Effect | No delivery of artifacts possible |
| Vulnerability | •None or few access restrictions<br>•Insecure artifact repository environment |
| Threat | Upload a huge amount of data |
| Effect | Server crash because low capacity |
| Vulnerability | •No checking of uploaded amount of data |

**Table 5.15.:** STRIDE analysis: Denial of service attack on the artifact repository

Table 5.16, Table 5.17, and Table 5.18 show the results of the vulnerability detection of the testing stage.

The type of tampering in the testing stage which are possible are presented in Table 5.16. Assuming that an attacker can deploy malicious code on the test server then this is possible because of none or few access restrictions or or because an unencrypted connection exists (e.g., HTTP). Tested artifacts are not stored again in the artifact repository after the testing stage. As a result, no damage is performed on the CD pipeline itself. If the attacker is able to manipulate testing results then it causes no damage on the CD pipeline itself. This action is also possible if the version of the testing server is vulnerable.

| Occurrence | Testing stage |
|---|---|
| Threat type | S **T** R I D E |
| Threat | Deploy malicious source code on test server |
| Effect | No damage effects on CD pipelines |
| Vulnerability | •None or few access restrictions<br>•Unencrypted connection (e.g., HTTP) |
| Threat | Attacker can manipulate test results |
| Effect | Modified testing results; no damage for CD pipeline |
| Vulnerability | •Vulnerable version of the testing server |

**Table 5.16.:** STRIDE analysis: Tampering in the testing stage

Table 5.17 shows the threats, effects, and vulnerabilities which can occur if in the pipeline testing stage information are disclosed. If an attacker can read the testing results then it causes no discernible damage to the CD pipeline itself. This can be possible

if the attacker can gain the access rights for the place where the testing results are stored or the connection at the time of transferring the results is unencrypted (e.g., HTTP).

| Occurrence | Testing stage |
|---|---|
| Threat type | S T R **I** D E |
| Threat | Read testing results |
| Effect | No damage effects on CD pipelines |
| Vulnerability | •None or few access restrictions<br>•Unencrypted connection (e.g., HTTP) |

**Table 5.17.:** STRIDE analysis: Information disclosure in the testing stage

A DoS attack could be a reason why no testing of the application is possible. The possible threats, effects, and vulnerabilities are presented in Table 5.18. The threat which could occur is that the testing server is shut down. If the testing server environment is insecure and there exists none or few access restrictions then an attacker can exploit these vulnerabilities.

| Occurrence | Testing stage |
|---|---|
| Threat type | S T R I **D** E |
| Threat | Shut down testing server |
| Effect | No testing of application is possible |
| Vulnerability | •None or few access restrictions<br>•Insecure testing server environment |

**Table 5.18.:** STRIDE analysis: Denial of service attack on the testing stage

Table 5.19, Table 5.20, and Table 5.21 show the results of the vulnerability detection in production stage.

Table 5.19 shows the threats, effects, and vulnerabilities which can occur if the pipeline production stage is tampered with. One threat is that an attacker can deploy artifacts on another server. As a result of this, unauthorized persons can gain access to these artifacts. If there exists an unencrypted connection (e.g., between the deployment server and the artifact repository) or a vulnerable version of the deployment server is in use then the attacker could exploit these vulnerabilities. Customer satisfaction would be reduced if there is no or malicious artifacts on the deployment server. This occurs if an attacker can exchange the artifacts on the server. This could happen if there exist none or few access restrictions. If the customer installs the malicious artifact and an attacker exploits the included vulnerabilities then it can damage the customer. Another

threat is that an attacker can change the server security configurations. The effect is that the server is opened to unauthorized persons. This would contribute to the loss of a company's image. An attacker can exploit this threat when the deployment server has low or none access rights.

| Occurrence | Production stage |
|---|---|
| Threat type | S **T** R I D E |
| Threat | Deploy artifact on another server |
| Effect | Unauthorized persons get access on artifacts |
| Vulnerability | •Unencrypted connection (e.g., HTTP)<br>•Vulnerable version of deployment server |
| Threat | Exchange artifacts |
| Effect | User uses malicious artifacts; no artifacts are available |
| Vulnerability | •None or few access restrictions |
| Threat | Change security configurations |
| Effect | Make deployment server accessible to unauthorized persons |
| Vulnerability | •None or few access restrictions |

**Table 5.19.:** STRIDE analysis: Tampering in the production stage

Table 5.20 shows the threats, effects, and vulnerabilities which can occur if in the pipeline production stage information are disclosed. Table 5.20 shows the threats, effects, and vulnerabilities if information disclosure is done in the production stage. If there exists an unencrypted connection (e.g., HTTP) then an attacker can read the deployed artifact. This attack would not lead to a damage to the CD pipeline.

| Occurrence | Production stage |
|---|---|
| Threat type | S T R **I** D E |
| Threat | Read deployed artifacts |
| Effect | No damage effects on CD pipelines |
| Vulnerability | •Unencrypted connection (e.g., HTTP) |

**Table 5.20.:** STRIDE analysis: Information disclosure in the production stage

Denial of Service attacks which are feasible in production stage are listed in Table 5.21. Insecure deployment server environment or none or few access restrictions to the server could allow an attacker to shut down the server. As a result of this, no artifact could go into production. This would dissatisfy the customer.

| Occurrence | Production stage |
|---|---|
| Threat type | S T R I **D** E |
| Threat | Shut down deployment server |
| Effect | No artifact can going into production |
| Vulnerability | •None or few access restrictions<br>•Insecure deployment server environment |

**Table 5.21.:** STRIDE analysis: Denial of service attack on the production stage

All in all, there are many entry points due to none or few access restrictions or unencrypted connections between two CD pipeline components.

## 5.3. Detection of vulnerabilities in Jenkins

Jenkins is the CI server employed in both investigated CD pipelines. If the correct circumstances exist, the following threats could occur: A change of the Jenkins configuration file without notification on the Jenkins UI and without Jenkins authorization, a change of security properties without notification on the Jenkins UI and a removal of files with Jenkins.

### 5.3.1. Change Jenkins configuration file without notification and Jenkins authorization

The vulnerability is to change the configuration file in the Docker environment without any notification on the Jenkins UI. The vulnerability is that the configuration file of Jenkins can be changed without any authorization and notification on the UI. The result of the exploit is that the Jenkins URL is open for all users who knows the URL. In addition, all users have admin rights and are able to do anything they want.

In Listing 5.1 the steps which need to be performed to exploit this vulnerability are listed. The precondition for the exploit is that you have access to the system where the Docker container is running. With the first command the Docker name is found out. The second to fourth commands help the person to find out the path where the config.xml is stored in the Docker. The next step is to copy the insecure config.xml (Appendix A) from outside into the configuration directory of the Docker.

The path to the Docker here is: *jenkins-master:/var/jenkins_home/config.xml*. The first part is the Docker name (here: jenkins-master ) and the second part is the path to the config.xml in the Docker. After that the Docker container has to be restarted

**Listing 5.1** Exploit vulnerability on Jenkins configuration

```
1   docker ps
2   docker exec jenkins-master
3   cd FIND PATH TO config.xml here: var/jenkins_home/config.xml
4   exit
5   docker cp config.xml jenkins-master:/var/jenkins_home/config.xml
6   docker restart jenkins-master
```

```
1   <authorizationStrategy
        class="hudson.security.AuthorizationStrategy$Unsecured"/>
2   <securityRealm class="hudson.security.SecurityRealm$None"/>
```

(see step 6). The two further lines show the contents of the config.xml which which effectively remove the CI server's security and make the CI server insecure.

### 5.3.2. Change Jenkins security properties without notification

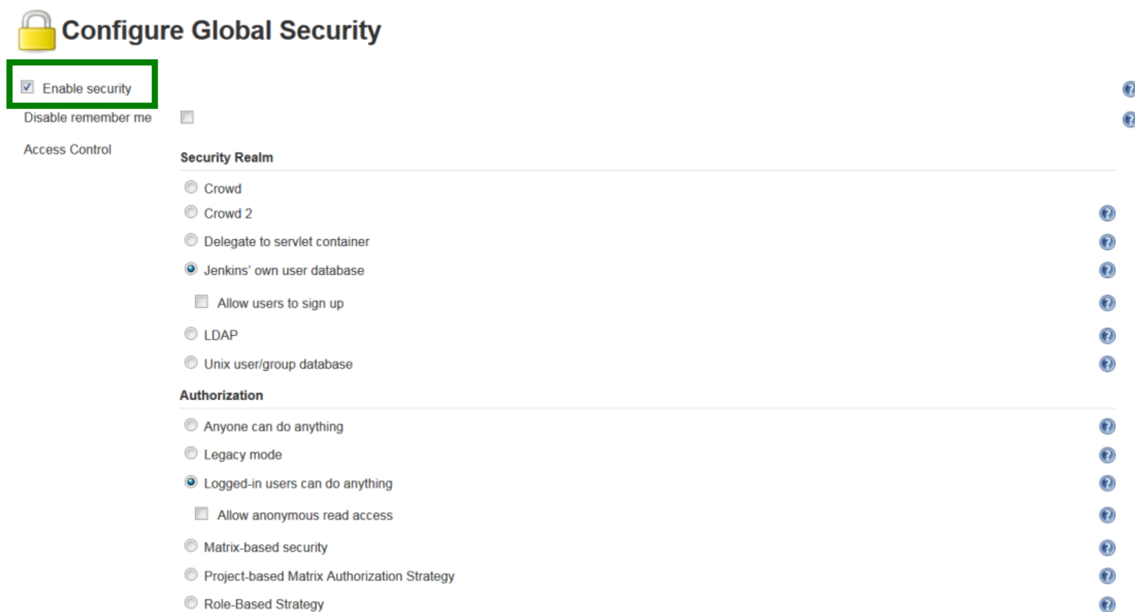



**Figure 5.2.:** Jenkins user interface: global security configurations

If persons have the rights to manage Jenkins and Jenkins is using its default configuration then it is possible to configure global security of Jenkins and to disable its security. This security configurations are shown in Figure 5.2. If the check mark is removed (green square) then the Jenkins UI is open to unauthorized persons. This is one possibility to change the security configuration options. Figure 5.2 shows the other options which are available for changing configuration security. On the one hand, the

**Listing 5.2** Remove file with integrated Jenkins shell command executor

```
1  ls -al /jenkins_home/
2  rm -rf /jenkins_home/removeFolder
3  ls -al /jenkins_home/
```

security realm can be defined, for example it can be restricted that not an unauthorized user can create an account. On the other hand, the privileges of the users can be restricted. If there is no plugin installed on Jenkins then the changes are not logged and nobody can trace back the changes. There is also no notification on the UI.

### 5.3.3. Remove relevant files with Jenkins

Jenkins permits the execution of arbitrary shell commands. The consequence of this is that it is possible to irreversibly remove folders or files. If the appropriate plugin is not installed there exists no notification which shows changes on pipeline jobs. Listing 5.2 lists commands which have to be executed to remove a folder. The requires steps for this action is that you have to install Jenkins in a Docker container and create a pipeline in Jenkins. When creating a project, select "execute shell" and insert the following lines of code (see Listing 5.2) into the field. Furthermore, a folder with the name "removeFolder" has to be created in the Docker container under the path */jenkins_home/*. After that, the pipeline has to be saved. In the next step the pipeline has to be built and the "removeFolder" is deleted. The first and third command only display the directory context before and after the folder deletion. As a result of this, every folder or relevant file (e.g., Jenkins config file) can be deleted. This can damage the CD pipeline or it is possible that the CI server crashes due to this attack.

## 5.4. Summary of vulnerabilities in CD pipelines

The results of the survey and this chapter summarize that CD pipelines may have the following vulnerabilities:

- Internal employees (human errors)
- Unencrypted connections between CD pipeline components
- Insecure environment of the CD pipeline components
- None or few access restrictions
- Use of vulnerable versions of the CD pipeline components

- Vulnerable CD pipeline configurations
- Vulnerable code commits, CD pipeline scripts, Docker images/containers, artifacts
- No review of changes on the CD pipeline

Chapter 6

# Tools for securing CD pipelines

If vulnerabilities should be continuously detected in a CD pipeline tools are needed. The first part of this chapter describes the research method for tool selection (Section 6.1). In the further sections such tools are listed and described in categories which can be used to detect vulnerabilities in CD pipelines. The selected tools can detect or mitigate the vulnerabilities in the following categories: CD pipeline files (Section 6.2), CD pipeline components (Section 6.3), sensitive data in commit messages (Section 6.4), CD pipeline configurations (Section 6.5), application dependencies (Section 6.6), Docker containers/images (Section 6.7), artifacts (Section 6.8), CD pipeline infrastructure (Section 6.9), and monitoring and logging tools (Section 6.10). The last part of this chapter provides a summary overview of tools mapped to the vulnerabilities and pipeline stages (Section 6.11).

## 6.1. Research method and literature foundations

According to Feiman and McDonald [FM09], there exists a demand for security tools and therefore a huge amount of tools are developed. Because of this fact, the search for tools is limited to free open source tools and commercial tools which are extensions of the tools already used in the investigated pipelines. Free open source tools are preferred because they incur no additional costs for companies. For the CI tools, the focus lies on Jenkins because according to CloudBees [Clo17] Jenkins has in 2017 "over 1,000,000 users worldwide [which] make Jenkins to the most widely-used, open source automation server". The survey also shows that all participants know or use this tool.

The selection methods for the tools are described in the following. First, a literature research in the context of DevSecOps is undertaken. The results of this research are the papers and presentations of Ullah et al. [URS+17], Schneider [Sch15], Storms [Sto15], Lipke [Lip17], Bird [Bir16], Shu et al. [SGE17], and Kuusela [Kuu17]. These works

form the basis for this chapter. These papers and presentations include tools which can be included in the CD process. In addition, they presented tools which can secure web applications. Besides those tools, they mentioned tools through which security tests for applications can be written.

Supplementing this research effort a further web search was carried out, in which additional tools were determined for the detected vulnerabilities in Chapter 5.

## 6.2. Detection of vulnerabilities in CD pipeline files

Changes in the CD pipeline files like the Jenkinsfile, travis.yml or concourse.yml can cause a failure of the CD pipeline. If the CD pipeline file has an error then it might be that no code change can be deployed. Therefore, it is necessary to secure the CD pipeline file. There exist several CI tools but not all were considered. The employees of the specific company were tested about their awareness of some CI tools. The result shows that there does not exist much knowledge about CI tools (see Figure 4.1), with the exception of all survey participants being familiar with Jenkins. The thesis will only focus on the top three CI tools of the survey result and which are open source. These tools are Jenkins, Concourse CI and Travis CI. In following subsections tools are presented which can detect or mitigate the vulnerabilities in these three CD pipelines files (Jenkinsfile, Travis.yml, and Concourse.yaml).

#### Jenkinsfile

Jenkins Pipeline Unit[1] is a testing framework to test Groovy DSL[2] pipeline syntax [oA] CD pipeline files which are written in Grooy DSL syntax can get very complex. The aim of the tool is to write "unit tests on the configuration and conditional logic of the pipeline code, by providing a mock execution of the pipeline" [oA]. The test can be written in Groovy or Java 8. Jenkins executes not directly Groovy but transfers the script files into the continuation-passing style (CPS). The test can be used to check if the Groovy script is serializable. Listing 6.1 shows an example unit test which detects if the execution of a Jenkinsfile runs without errors. The Jenkinsfile is written in Groovy syntax. In the test the Jenkinsfile has to be loaded and afterward executed (line 9 and 10). In addition to that, there exists the project pipelineUnit[3] which shows how to write unit tests for

[1]https://github.com/jenkinsci/JenkinsPipelineUnit
[2]https://jenkins.io/doc/book/pipeline/jenkinsfile/
[3]https://github.com/macg33zr/pipelineUnit

**Listing 6.1** Jenkins Pipeline Unit: Example unit test [oA]

```
1 import com.lesfurets.jenkins.unit.BasePipelineTest
2
3 class TestExampleJob extends BasePipelineTest {
4
5 //...
6
7 @Test
8 void should_execute_without_errors() throws Exception {
9 def script = loadScript("job/exampleJob.jenkins")
10 script.execute()
11 printCallStack()
12 }
13 }
```

declarative pipelines and scripted pipelines. This project includes the Jenkins Pipeline Framework.

### Travis.yml

In the tool Travis CI the pipeline is declared in a travis.yml file. Travis CI includes a YAML validator[4]. If the file is validated before committing then it can reduce the following build errors:

- Invalid YAML file
- Missing language key
- Unsupported runtime versions of Ruby[5] or PHP[6] for example
- Deprecated features or runtime aliases

The validation can be executed from the command line.

### Concourse.yaml

Similar to Travis Concourse CI has also included a pipeline validator[7]. This pipeline script validation is used to detect errors before they are committed.

[4]https://docs.travis-ci.com/user/travis-lint
[5]https://www.ruby-lang.org/en/
[6]http://php.net/
[7]https://concourse.ci/fly-validate-pipeline.html

## 6.3. Detection of vulnerabilities in CD pipeline components

If the components of the CD pipeline has vulnerabilities then an attacker is possible to exploit them. As a result of this, CD pipelines can be manipulated. Therefore, it is necessary to check if CD pipeline components have vulnerabilities.

**CIRCL CVE Search**

CIRCL CVE Search[8] is an HTTP API (application programming interface) which can detect vulnerabilities in CD pipeline components. The results of a search with the API are printed in a JSON (JavaScript Object Notation)[9] file. An example search which outputs the vulnerabilities in Jenkins is called as follow:

curl http://cve.circl.lu/api/browse/Jenkins

## 6.4. Detection of sensitive data in commit messages

The following section lists tools which can detect sensitive data in commit messages.

**truffleHog**

truffleHog[10] searches for sensitive data in git repositories. This tool has the ability to search in the entire commit history. As a result of this, sensitive data which are already committed can also be detected. There are two possibilities to execute this tool. Either it is done manually or a script is written which includes the execution command. This script can be integrated in a Jenkinsfile for example and so it is possible to integrate this tool execution into the CD process. truffleHog can prevent the employees of committing sensitive data in the source code and the commit messages. It is possible that the tool detects false positive results. If false positive results occur then the developer has manually to decide which result is correct.

[8] https://www.circl.lu/services/cve-search/
[9] https://www.json.org/json-de.html
[10] https://github.com/dxa4481/truffleHog

### git secrets

git secrets[11] is another tool to detect sensitive data in the commit messages and the source code. Similar to truffleHog it can scan in the entire commit history. This tool prevents the employee from committing passwords and sensitive information to a git repository. For each sensitive information, a pattern has to be added to the tool. A negative aspect is that added patterns cannot be removed with a command. It is possible that the tool detects false positive results. If false positive results occur than the developer has manually to decide which result is correct.

### GitHub GPG signature verification

GitHub GPG signature verification[12] can be used to show if the commits are from a trusted source. The function to verify a commit signature is included in git[13]. The GNU Privacy Guard (GPG)[14] can be used to sign commits. Every developer who wants to do a commit has to upload a GPG key to the source code repository.

## 6.5. Detection of vulnerabilities in pipeline configurations

Pipeline components should be secured against access by unauthorized persons. If Jenkins is installed with default configuration, every Jenkins user has administrator permission[15]. On the Jenkins UI with one simple click, the whole global security settings can be turned off. By default, no changes of the security configurations are automatically logged. Thus, there exists no traceability of the user who has done these changes.

### JobConfigHistory plugin

The JobConfigHistory[16] is a plugin for Jenkins that shows which configuration and job changes are made by which user. On the Jenkins UI the plugin shows the time and the user who has performed the changes. Figure 6.1 shows an excerpt of the plugin output on the Jenkins UI. It can be seen that a user has changed the configuration file on a

[11] https://github.com/awslabs/git-secrets
[12] https://blog.github.com/2016-04-05-gpg-signature-verification/
[13] https://git-scm.com/book/en/v2/Git-Tools-Signing-Your-Work
[14] https://www.gnupg.org/
[15] https://wiki.jenkins.io/display/JENKINS/Standard+Security+Setup
[16] https://wiki.jenkins.io/display/JENKINS/JobConfigHistory+Plugin

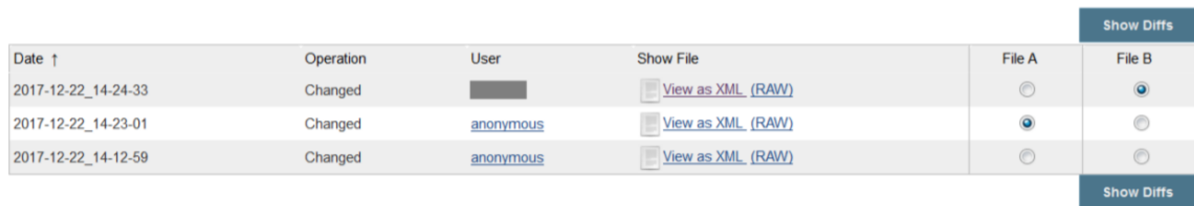



**Figure 6.1.:** Output of the JobConfigHistory plugin

specific date. In addition, the changes which were done can also be displayed on the Jenkins UI.

### SCM Sync configuration plugin

The SCM Sync configuration plugin[17] is another plugin for Jenkins. The feature of the plugin is that it synchronizes the configuration files with a repository. As a result of this, changes are committed and can be traced back. The plugin helps developers to detect who changed the configurations files and what changes were made (e.g., job and global configuration). It hasn't been developed since a year and the synchronization repository can only be a public one. The tool can be integrated into the CD pipeline but because of aforementioned limitations it not possible to use it in the case study.

### Audit trail plugin

The audit trail plugin[18] for Jenkins is a tool that logs the changes on creation of jobs or traces which person has started or removed the build. The tool saw its last modification three years ago. Because the tool is not up-to-date, it is not used in the case study.

### Jenkins matrix-based security

The matrix-based security[19] property limits the privileges of the user and groups of Jenkins. These matrix properties can increase the security level of Jenkins because not

[17] https://github.com/jenkinsci/scm-sync-configuration-plugin
[18] https://github.com/jenkinsci/audit-trail-plugin, https://plugins.jenkins.io/audit-trail
[19] https://wiki.jenkins.io/display/JENKINS/Matrix-based+security

every developer can do everything. This can mean that not all developers can change configuration files and pipeline jobs.

#### Travis CI system info

The system info feature[20] of the CI tool Travis CI can log changes on the configuration files.

#### Security Monkey

AWS accounts can be analyzed and monitored through the tool Security Monkey[21]. The tool tracks changes on the policy of the accounts and checks for insecure configurations. As a result of these security-related changes and configurations in the AWS environments are logged.

## 6.6. Detection of vulnerabilities in application dependencies

According to the OWASP Top 10 list, which was published 2017, components with known vulnerabilities are a high risk [OWA17b]. The following section lists tools which are able to detect vulnerabilities in foreign components of the applications.

#### OSSIndex

OSS Index[22] (open index of open source) provides information about vulnerabilities. There exists a REST (representational state transfer) API which is used by some open source tools such as Auditjs[23] (an NPM project), ossindex-maven-plugin[24] and ossindex-gradle-plugin[25].

[20] https://github.com/travis-ci/system-info
[21] https://github.com/Netflix/security_monkey
[22] https://ossindex.net/
[23] https://github.com/OSSIndex/auditjs
[24] https://github.com/OSSIndex/ossindex-maven-plugin
[25] https://github.com/OSSIndex/ossindex-gradle-plugin

Auditjs uses the OSS Index v2 REST API to identify known vulnerabilities. Developers can create whitelists so that not all detected vulnerabilities break the build.

The oss-index maven and gradle plugins can be used to detect vulnerabilities in the foreign used libraries.

#### OWASP dependency check

OWASP dependency check[26] is used to identify vulnerabilities in application libraries. The plugin can be integrated in Jenkins. OWASP dependency check uses the NVD and CVE databases to detect the vulnerabilities in the libraries. The tool can output false positive results, which means that libraries in which vulnerabilities are detected are not used. If the Common Platform Enumeration (CPE) confidence of the library is low, then these results can be eliminated because they are false positives.

#### Snyk

Snyk[27] is an online platform which detects vulnerabilities in dependencies. Snyk uses vulnerability databases such as CVE. Furthermore, it monitors GitHub users commit messages which may contain security-critical information. Snyk supports projects in various programming language for example Java, Node.js or Python. The repository has to be connected with the Snyk platform. One advantage of this tool is that the repository is checked for vulnerabilities in regular intervals and a report is published on the UI and can be shared via e-Mail. The disadvantage is that Snyk needs the credentials of the repository and saves the data on their servers.

#### Retire.js

The Retire.js[28] tool supports the vulnerability detection in used JavaScript libraries. Web application can be inspected with this tool.

#### Gradle Witness

Remote dependencies of the application are stored in repositories like Maven Central. These dependencies can be .jar or .aar (Android Archive) files and are stored with their

[26] https://www.owasp.org/index.php/OWASP_Dependency_Check
[27] https://snyk.io/docs/security
[28] https://github.com/RetireJS/retire.js

calculated checksums md5sum and sha1sum. If a project is built through gradle then the used dependencies are downloaded from the remote repository. In addition to that, the checksum files are also downloaded. If an attacker has the ability to replace the libraries and the checksum then the malicious libraries are downloaded and built without any test.

The Gradle Witness[29] tool is able to prevent this and can detect such vulnerability. This is possible because Gradle Witness verifies the downloaded checksum with the used checksum. If both are not equal then the build is aborted. The integrity of the application dependencies is verified by this tool.

## 6.7. Detection of vulnerabilities in Docker containers/images

According to Delafuente [Del15], it is important to know how secure the Docker containers are which the company uses. He published 10 possible tools which can be used to detect vulnerabilities in a container. In the following section the tools of the list are shown which can be used to detect vulnerabilities in CD pipelines. Additionally the list also includes tools which were found out by doing a web search.

#### Docker Hub image security scan

Docker Hub[31] is a registry for Docker images. Docker Hub has included an image security scanner[32] which can detect vulnerabilities in Docker official images. For official images only the latest versions of images are automatically scanned. The results of the Docker security scanning service for the official Jenkins image is presented in Figure 6.2. These scan results are from the 9th of April 2018. The results show that the official Jenkins image has vulnerabilities. A Docker Hub account is needed to see the results. For private repositories the service ended on 31st March 2018.

[29]https://github.com/signalapp/gradle-witness
[31]https://hub.docker.com/
[32]https://docs.docker.com/datacenter/dtr/2.4/guides/user/manage-images/scan-images-for-vulnerabilities/,https://docs.docker.com/docker-hub/official_repos/

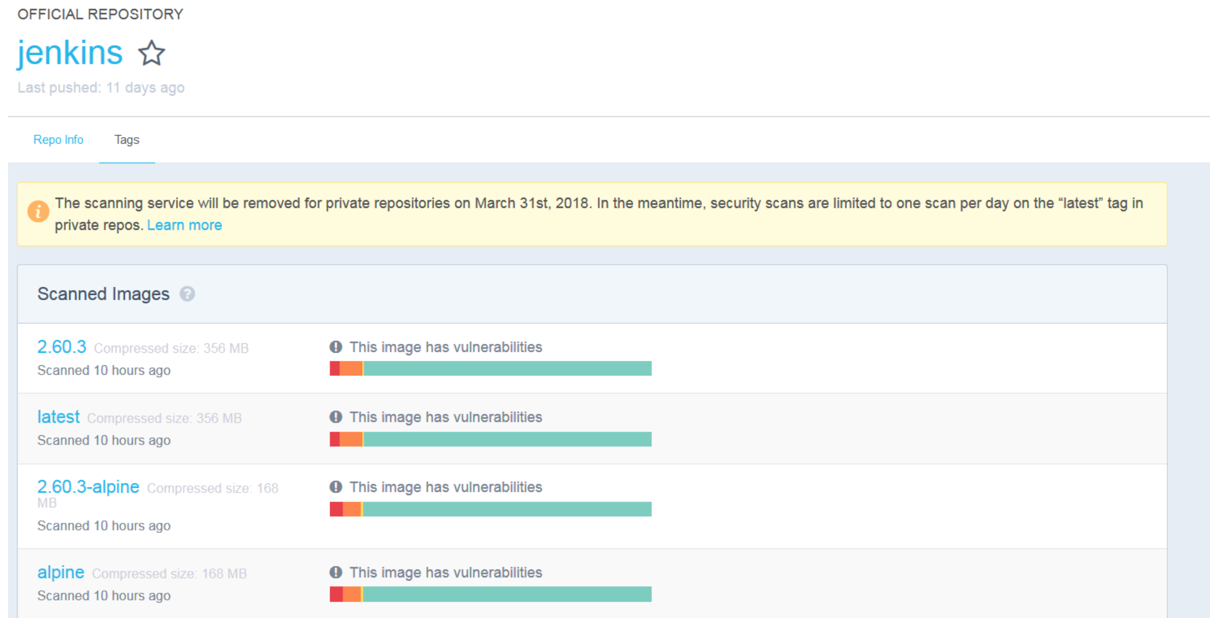



**Figure 6.2.:** Docker Hub image security scanning results of official Jenkins image[30]

### Open Security Content Automation Protocol (OpenSCAP)

The open source tool OpenSCAP[33] is a security compliance solution. With this tool images, containers and virtual machines can be scanned. OpenSCAP has a graphical user interface on which the detected vulnerabilities are shown. The tool is based on the NVD database. The disadvantage of this tool is that only Red Hat Enterprise Linux (RHEL) images and containers can be scanned. Those containers are stored in the Red Hat Container Catalog[34].

### Dagda

With Dagda[35] vulnerabilities can be detected in Docker images and containers. The detection is performed with a static analysis method. For the detection the CVE database has to be downloaded first. The OWASP dependency check and the Retrie.js are the basis for this tool.

[33]https://www.open-scap.org/
[34]https://access.redhat.com/containers/
[35]https://github.com/eliasgranderubio/dagda

### CoreOS Clair

CoreOS Clair[36] is a tool which can detect vulnerabilities in images and containers. Clair can detect vulnerabilities in Docker images which are stored in the Amazon EC2 registry. The additional tool Clairctl can be used for analyzing and making HTML-reports[37]. According to Delafuente [Del15], the installation is difficult and the usability lacks.

### Jenkins CI Image Vulnerability Scan

Jenkins CI Image Vulnerability Scan is a tool which is able to detect vulnerabilities in built Docker images with CoreOS Clair in Jenkins.

### Banyan Collector

Banyan Collector[38] is a framework which examines images. For this tool the programming language go[39] has to be installed and the collector has to be run on a Docker Host. As of writing this thesis, this tool is obsolete because it has not been maintained for a year.

### Twistlock

Twistlock[40] is a commercial tool which is able to detect vulnerabilities in Docker containers. The tool can be integrated into CI/CD tools and is able to scan any image registries. Twistlock avoids the deployment of images which have vulnerabilities.

### Black Duck Hub

Applications and containers oftentimes depend on many open source components. Every year 4000 new vulnerabilities in open source software occur. Black Duck Hub[41] is a commercial tool that automates and manages open source security. Black Duck Hub continuously detects vulnerabilities in open source components. The database is

[36] https://github.com/coreos/clair
[37] https://github.com/jgsqware/clairctl
[38] https://github.com/banyanops/collector
[39] https://golang.org/doc/install
[40] https://www.twistlock.com/get-twistlock/
[41] https://www.blackducksoftware.com/products/hub

based on VulDB and NVD and through that the tool receives information about new vulnerabilities. The tool alerts the developer if there is a new vulnerability in the open source components. In addition, Black Duck Hub can scan Docker containers and decides if they are free of open source vulnerabilities. The tool can detect vulnerabilities in containers which are stored in the Amazon EC2. Furthermore, Black Duck Hub can be integrated in Jenkins.

## 6.8. Detection of vulnerabilities in artifacts

The following section lists tools which are able to detect vulnerabilities in artifacts. The detection of vulnerabilities in artifacts is useful because otherwise malicious or vulnerable artifacts are going into production. Because of the fact that the production stage is part of a CD pipeline, it is necessary to know that the artifacts are free of vulnerabilities otherwise an attacker is able to harm the user.

**JFrog Xray**

JFrog Xray [JFr] is a commercial continuous security and universal artifact analysis tool which can scan vulnerabilities in Docker images and artifacts (e.g., .jar files). The tool can scan the following types of packages [JFr]:

- Java (Maven)
- JavaScript (NPM, Bower)
- .NET (Nuget)
- Python (PyPi)
- Docker (Debian/RPM)

At the moment this scanner is the only tool which can be integrated with the JFrog Artifactory [JFr]. As of writing this thesis there was no support for gradle packages.

**Nexus Repository Manager OSS Repository Health Check**

The Health Check[42] is a feature of the Nexus Repository Manager. The Health Check analyzes the components of the repository and detects their vulnerabilities in it. The tool

[42]https://blog.sonatype.com/how-to-use-the-new-repository-health-check-2.0

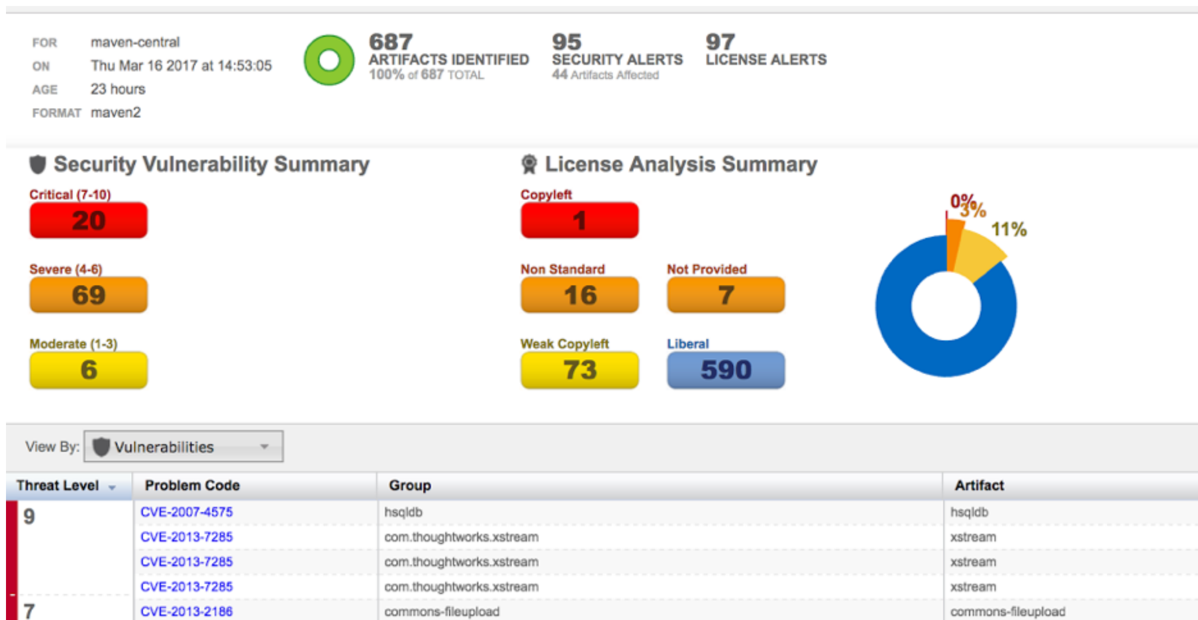



**Figure 6.3.:** Example results of Nexus Repository Manager Pro Health Check

supports a various amount of archives formats such as .jar, .zip, .exe, .apk. If a company uses the Nexus Repository OSS 2.0 then only the top five vulnerable dependencies are shown. A hint is given how to mitigate them. If a company uses the Nexus Repository OSS 3.0, the report is a security vulnerability summary of all detected vulnerabilities. This means that only an overview is given about the categories of the detected vulnerabilities. Through the report, for example, it can be seen that 20 vulnerabilities are critical. The report does not provide information about the artifact in which the vulnerabilities occur. But only if a company uses the Nexus Repository Manager Pro an output as illustrated in Figure 6.3 is given. Only in this report the developer knows which artifact has vulnerabilities. The report shows alerts for the detected vulnerabilities. In addition, it is displayed which vulnerability is detected. The tool scans for vulnerabilities based on information from the CVE database.

## 6.9. Detection of vulnerabilities in infrastructure

If the infrastructure (environment of the CD pipeline components) has vulnerabilities then an attacker has the capability to damage the entire pipeline because he would be able to shut down one of its component. In the following paragraphs those tools are listed which can detect vulnerabilities in the infrastructure.

**DevSec Hardening Framework**

The DevSec Hardening Framework [Deu] describes itself as a tool which "[...] combines DevOps with security by adding a security layer into [an] automation framework, that configures [...] operating systems and services. The tool takes care of difficult settings, compliance guidelines, cryptography recommendations, and secure defaults" [Deu]. Multiple cookbooks are available for Chef, Ansible, and Puppet which can be applied to the running machine. As an example, the ansible-os-hardening cookbook is based on security guidelines of Ubuntu[43], Deutsche Telekom[44], Arch Linux[45], and National Security Agency (NSA)[46]. The tool can be used to secure the environment of the CD pipeline components.

**Lynis**

Lynis[47] is a tool which can detect vulnerabilities in Linux and Unix-based systems. A further goal of Lynis is to harden such systems. For testing various kinds of sources are used such as OpenSCAP data[48].

**AWS Cloud Watch**

For AWS there is the tool Cloud Watch[49] which monitors the resources and applications which running on AWS. Cloud Watch has the option to to set alarms when something unfamiliar happens.

## 6.10. Detection of vulnerabilities with monitoring and logging tools

Monitoring and logging tools are useful to detect vulnerabilities in each component of the pipeline because they are able to alert a developer if something unexpected happens.

[43] https://wiki.ubuntu.com/Security/Features
[44] https://www.telekom.com/de/verantwortung/datenschutz-und-datensicherheit/sicherheit/sicherheit/privacy-and-security-assessment-verfahren-342724
[45] https://wiki.archlinux.org/index.php/Sysctl
[46] https://www.nsa.gov/what-we-do/information-assurance/
[47] https://cisofy.com/lynis/
[48] https://www.open-scap.org/features/standards/
[49] https://blog.novatec-gmbh.de/check-your-logs-with-cloudwatch/

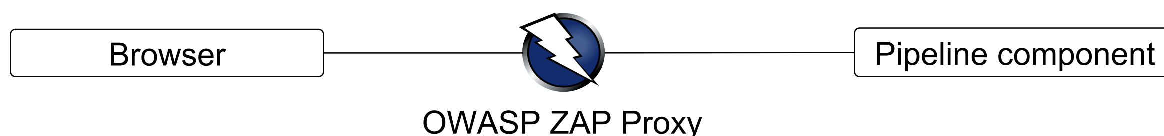



**Figure 6.4.:** Insertion of OWASP ZAP Proxy in the CD process based on [OWAc]

Some of the tools listed below are able to detect vulnerabilities in the connection between two pipeline components. Other tools which are listed below can detect when an unauthorized or authorized user logs into the CD pipeline components.

#### OWASP Zap Proxy

The OWASP Zap Proxy[50] is a penetration testing tool. One feature of the tool is that it can intercept network traffic. All request and responses have to pass the OWASP Zap Proxy. As seen in Figure 6.4, all requests and responses between the browser and the pipeline component are going through the OWASP Zap Proxy. If the proxy is enabled then it is possible to see and even manipulate the network traffic in OWASP Zap Proxy tool. If the connection uses the HTTPS protocol then depending on the application the OWASP Zap Proxy Certificate has to be installed in a browser or in an operating system. With this tool the network traffic, CPUs and file I/Os can be monitored or manipulated, for example, every HTTP request.

#### Elastic Logstash

Elastic Logstash[51] is an open source tool which can process and transform the data from different kinds of sources. Any type of log can be stored with these tools and an overview of the state of the CD pipeline components can be given.

#### Sysdig falco

Sysdig falco[52] is a monitoring tool for container and application security. With this tool containers, applications, and network traffic can be monitored, for example, each HTTP request of a container.

[50] https://github.com/zaproxy/zaproxy
[51] https://www.elastic.co/products/logstash
[52] https://www.sysdig.org/falco/, https://github.com/draios/falco

### Splunk

The tool Splunk[53] is an automated security monitoring system. There is a free version for a single user license with limited features. The commercial version can monitor and raise alerts. As an example, if a user tries often to log into the component and if it is set up Splunk can notify an employee that someone tries often to log into the component. Splunk can receive the data from different sources and displays a real-time visualization.

### OWASP Security Logging Project

The OWASP Security Logging Project[54] is an API for logging security events. The API extends the API Log4J[55] which is familiar to developers. The developers select this API if they consider security aspects. As an example, the API can be used to log security events (e.g., user log in/out, successful/failed login, password changes), to show the history of security events, or to track sensitive information (e.g., username, event type)[DKS16].

## 6.11. Summary overview of the selected tools and detected vulnerabilities

The following two tables Table 6.1 and Table 6.2 give a summary of the selected tools, the vulnerabilities they are able to detect and in which pipeline stage they can be included. In addition, it is presented if the tool is a detection or a mitigation tool.

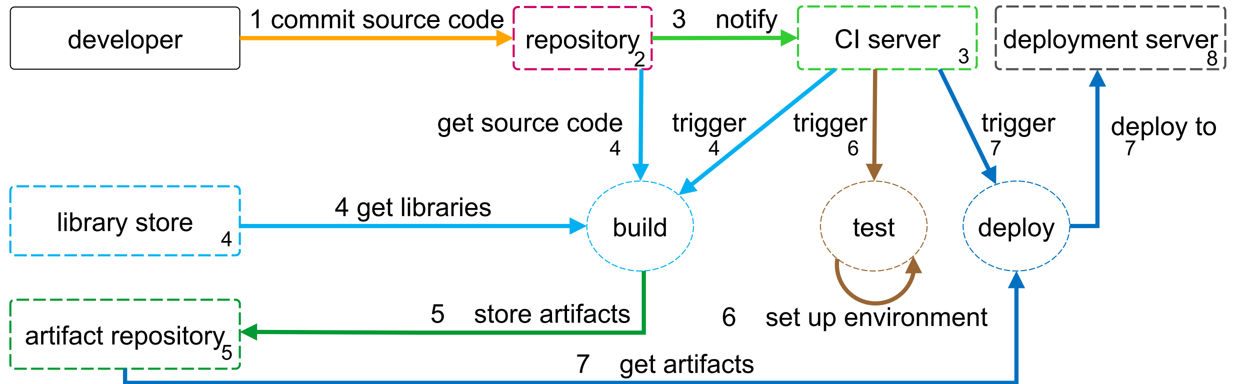
Table 6.1 presents those tools which can be used for one specific CD pipeline component or stage. It can be seen that there are stages such as the production stage (see step 7 and 8) where no specific tools are available. In the repository, internal features can be used which means that another developer has to review the code changes.

Table 6.2 presents those tools which can be used in more than one stage and for more components. It can be seen that for every CD pipeline stage and component a tool with specific features is available.

Most of the dependency check tools are only necessary if the security state of the application would be tested. Only the tool Gradle Witness tests the checksum of the downloaded library and checks if it is from the right source. In case study the other

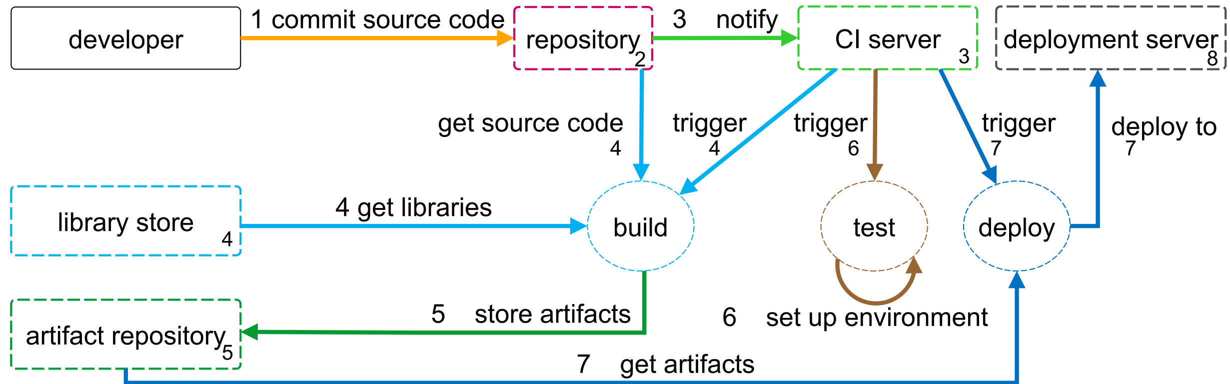
[53]https://www.splunk.com/
[54]https://www.owasp.org/index.php/OWASP_Security_Logging_Project,
https://github.com/javabeanz/owasp-security-logging
[55]https://logging.apache.org/log4j/2.x/

dependency are only used if there exists no artifact scanner which guarantee that in the production stage no vulnerability is included in the artifact used dependencies can be exploited to harm the user. To sum up, there exist several tools which can be integrated into the CD pipeline to increase its security level. Because of the fact, that each pipeline is different an analysis is necessary to be done to evaluate which tools can be used for the case at hand.




| Pipeline stage | Vulnerability | Tool | Detection (D), Mitigation (M) |
|---|---|---|---|
| Commit stage | No review of commit messages | git secrets | D |
| | | truffleHog | D |
| | | GitHub GPG signature verification | D, M |
| | No testing of pipeline scripts | Jenkins Pipeline Unit | D |
| | | Travis CI | D |
| | | Concourse CI | D |
| Build stage | No library checking | OWASP dependency check | D |
| | | ossindex-maven-plugin | D |
| | | ossindex-gradle-plugin | D |
| | | Auditjs | D |
| | | Snyk | D |
| | | Retire.js | D |
| | | Gradle Witness | D |
| CI Server (Jenkins) | None or few access restrictions | JobConfigHistory plugin | D |
| | | Jenkins matrix-based security | M |
| Artifact repository | Vulnerable artifacts | JFrog Xray | D |
| | | Nexus Repository OSS | D |

**Table 6.1.:** Overview: Tools which detect vulnerabilities in a single pipeline stage




| Pipeline stage | Vulnerability | Tool | Detection (D), Mitigation (M) |
|---|---|---|---|
| All components (2, 3, 5, 6, 8 ) | Vulnerable version of components | CIRCL CVE Search | D |
| | | Splunk | D |
| | | Elastic Logstash | D |
| | Vulnerable pipeline configurations | SCM Sync configuration plugin | D |
| | | Audit trail plugin | D |
| | | Travis CI system info | D |
| | | Security Monkey | D |
| | Vulnerable Docker images/containers | OpenSCAP | D |
| | | Dagda | D |
| | | Docker Hub image security scan | D |
| | | CoreOS Clair | D |
| | | Jenkins CI Image Vulnerability Scan | D |
| | | Banyan Collector | D |
| | | Twistlock | D |
| | | Black Duck Hub | D |
| | | Sysdig falco | D |
| | None or few access restrictions | OWASP security logging project | D |
| All stages (1, 2, 3, 4, 5, 6, 7, 8 ) | Insecure connection | OWASP Zap Proxy | D |
| | Insecure environment of the pipeline | DevSec Hardening Framework | M |
| | | Lynis | D |
| (3, 4, 5, 6, 7, 8 ) | Insecure environment of the pipeline components | AWS Cloud Watch | D |

**Table 6.2.:** Overview: Tools which detect vulnerabilities in multiple pipeline stages

Chapter 7

# Case Study

A case study was conducted to assess how secure industrial CD pipelines are and what security objectives are actually implemented to increase the level of security. The results of this case study show how some selected tools (see Chapter 6) can be integrated into a CD pipeline which is in operation. Additionally, the results of the tools should be interpretable and provide a value for the company.

This chapter has the following structure. In the first part it is described how the case study was executed (Section 7.1). The next two sections show the actual state of security of two CD pipelines (Section 7.2, Section 7.3). These two CD pipelines - in the following referred to as A and B - are investigated. Each investigation examines the CD pipeline setups, the use of security objectives and the assessment of the CD pipeline security levels by the team members. To show the employees the actual existent security level of their CD pipelines, selected tools are tested on them. The results are categorized according to the overall risk severity. At the end of each investigation, an overview of the entire security level of the CD pipeline is given. In a further part, one proof of concept is given (Section 7.4). A discussion about the investigations of both CD pipelines concludes the chapter (Section 7.5).

## 7.1. Case study design and planning

According to Runeson et al. [RHRR12], a case study is a research strategy which studies a specific case. In software engineering it is used to improve the software process or to get a deeper understanding of a phenomenon [RHRR12]. This case study is conducted to gain an understanding of the security levels and goals of industrial CD pipelines. In the following, the design and planning of the case study are described. As reported by Runeson et al., the design and planning of a software engineering case study which statistically investigates no hypothesis have the six following briefly described parts.

### 7.1.1. Rationale for the study

The case study is done on two CD pipelines A and B of one selected software consulting company. The reason for this case study is that developers can recognize which components and stages of their CD pipeline can be improved to increase the security level of it.

### 7.1.2. Objective of the study

The objective of the study is to answer the research question $RQ_6$ (Section 1.2). Therefore, it is investigated how secure the selected company specific CD pipelines A and B are. To achieve this aim vulnerabilities in these CD pipelines are detected with selected tools. The results should show that the security level in industrial CD pipelines can be improved because the company is only occasionally focusing on these security aspects.

### 7.1.3. Cases and units of analysis

Components and data flows of the two investigated CD pipelines A and B are described below. The structures of the CD pipelines were identified by oral interviews with the developers.

#### CD pipeline A

In Figure 7.1 the rough structure of the investigated CD pipeline A is presented. Until at the time of investigation the components of the CD pipeline are:

- Repository: Bitbucket 4.14.2
- CI server: Jenkins 2.89.3 and AWS
- Artifact repository: JFrog Artifactory 5.8.3
- Testing server: Jenkins and AWS nodes
- Deployment server: Rundeck 2.10.2-1 and HockeyApp store
- Other components: Puppet

The components are located in different places. That is why there exist data transfer boundaries. The Jenkins master is an internal server of the company. The Jenkins URL can only be accessed from the company's intranet. In an external cloud foundry

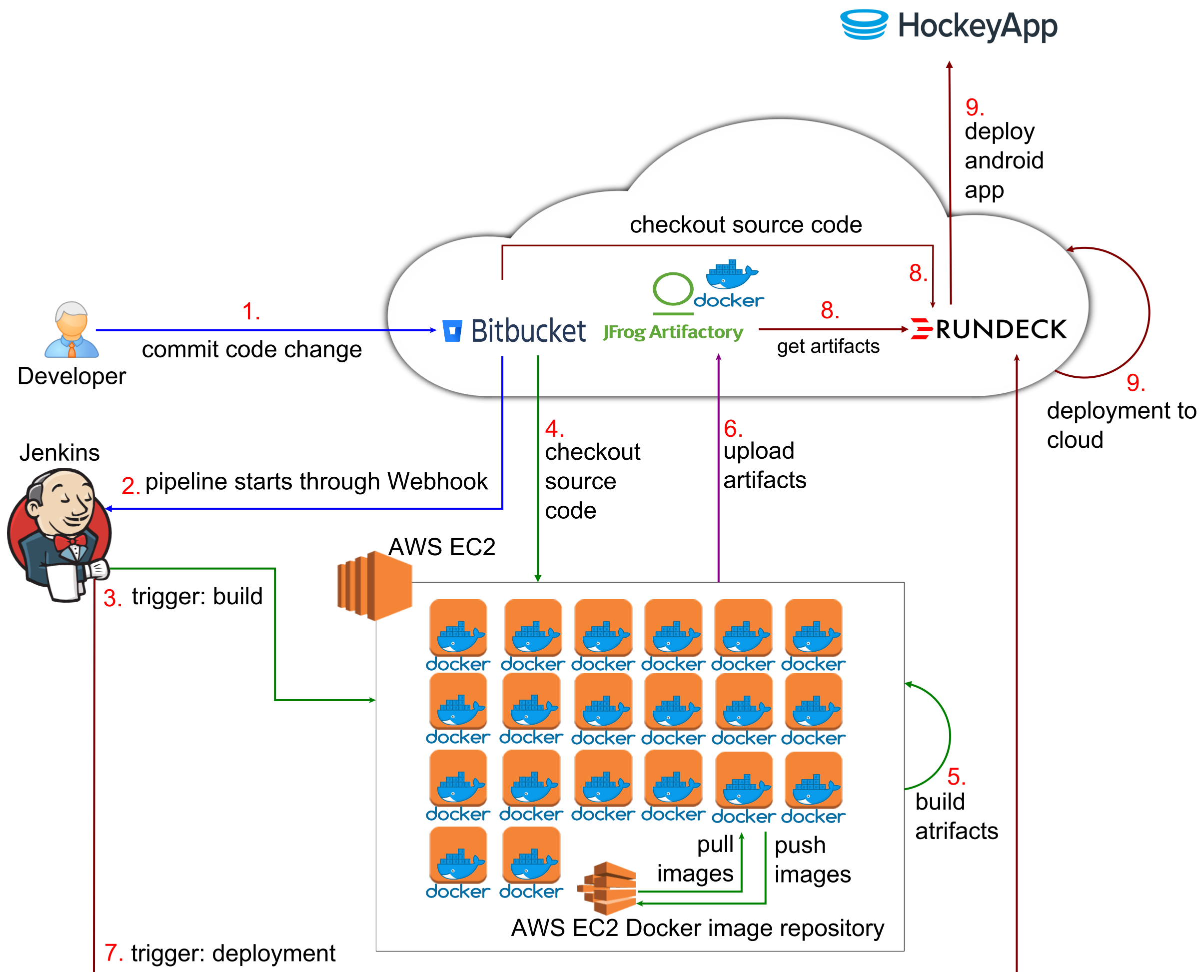



**Figure 7.1.:** Structure of the investigated CD pipeline A

the components Bitbucket, JFrog Artifactory and a Rundeck server are installed. For the build step AWS resources are used and for the deployment the HockeyApp store is needed. Puppet is used to set up the deployment environment.

In the following the CD pipeline A process is described.

### Commit stage and repository

At first a developer has to commit a code change to Bitbucket (see step 1. and 2. Figure 7.1). Another developer has to review the code before it can be committed into the repository. After the developer checked that the code has no issues, it can be committed. The Jenkins webhook checks if there is a change in Bitbucket. If the investigation leads to a positive result, Jenkins triggers a new instance of the CD pipeline.

**Build stage and CI server**

Jenkins checks out the source code from Bitbucket and triggers the build process (see step 3., 4., and 5. Figure 7.1). In this project every subproject (like the android app or the web user interface) is built on one of the 20 available Amazon EC2 container instances in the AWS cloud. In addition, the images for the container are stored in an Amazon EC2 Docker image repository which the AWS cloud provides. Before starting the build, the appropriate container image must be pulled from the AWS repository. The prerequisite for this is that a Jenkins job has loaded these necessary images into the AWS repository. The build phase is successfully completed when all JUnit tests have been passed and the static code analysis tools FindBugs, FindBugsSecurity, Checkstyle and SonarQube find no defects, errors or vulnerabilities. A successful build executes the next step of the CD pipeline (see purple lines in Figure 7.1), which means that the AWS cloud loads the artifacts into the JFrog Artifactory Repository.

**Artifact repository**

The product of the build stage is an artifact with is stored in the JFrog Artifactory (see step 6. Figure 7.1). Every version of the produced artifacts is stored for a certain time.

**Testing stage**

For every testing type a separate pipeline is triggered. If the described pipeline runs successfully, the automatic acceptance test (AAT) pipeline is started by Jenkins. The pipeline goes through the same steps as the first pipeline. Instead of building an artifact, the AAT pipeline performs all acceptance tests on the Amazon EC2 container instances and two Jenkins nodes. A successful execution of the AAT pipeline may trigger the load testing (LT) pipeline. But this phase only runs every night and not after every commit. The reason for this is that the test run is too long. Development can be impeded by the long duration of the tests.

**Production stage**

The deployment phase is triggered manually by a developer through the Jenkins user interface. Jenkins tells the Rundeck server which deployment phase it should perform. There exist several deployment environments with different purposes. During the sprints all development artifacts are deployed on the “dev” (development environment). Furthermore, there exists for example the production or review deployment environment. The aim of the deployment phase is to load the desired artifacts from the JFrog Artifactory and deploy them to the cloud foundry or, if it is an Android app, to the HockeyApp store (see step 7., 8., and 9. Figure 7.1). To do this, a gradle file must be loaded, which provides the artifact in the corresponding environment. If the Rundeck server should

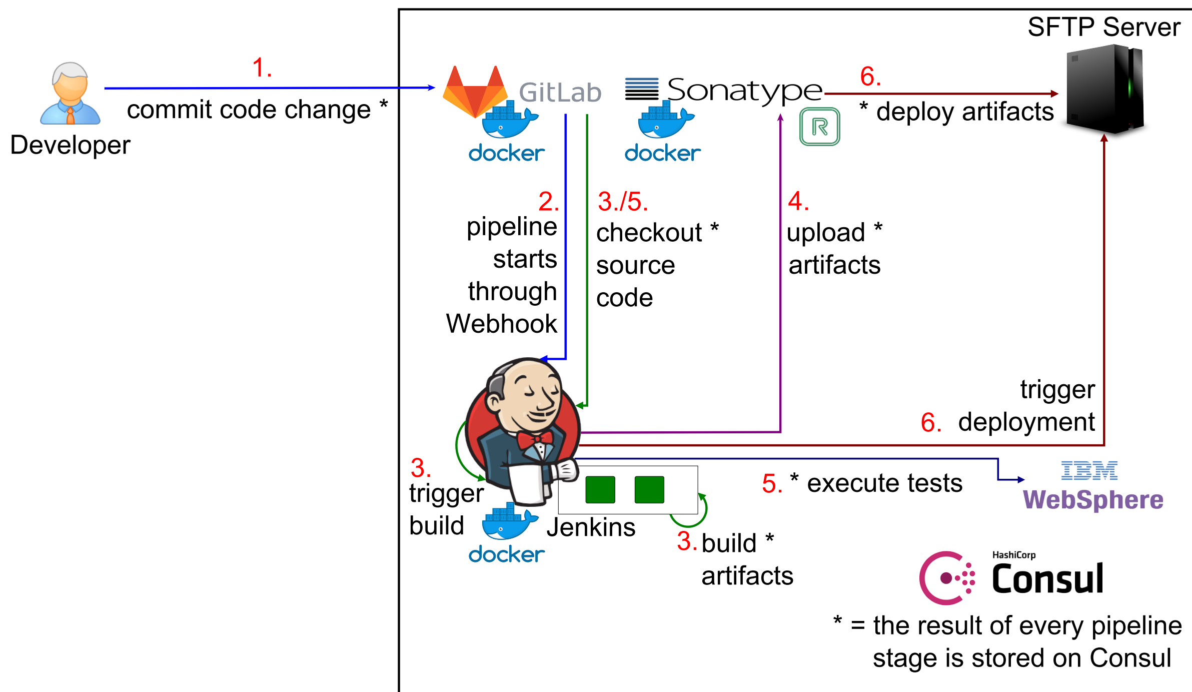



**Figure 7.2.:** Structure of the investigated CD pipeline B

deploy the artifact then it has to check out the deployment script which is stored in the Bitbucket repository. In this case, it checks out the entire source code. After the successful execution of this phase, the developer can check out the code changes on a browser or Android device.

### CD pipeline B

In Figure 7.2 the rough structure of the investigated CD pipeline B is presented. Until at the time of the investigation the components of the CD pipeline are:

- Repository: GitLab Community 10.1.0
- CI server: Jenkins 2.32.2
- Artifact repository: Sonatype Nexus Repository OSS 2.14.3
- Application server for the testing stage: IBM WebSphere 8.0.0.10
- Deployment server: SFTP server
- Other components: HashiCorp Consul 0.75.5

All components of the CD pipeline will be installed on the customer's infrastructure. This has the effect that the company is dependent on it. The customer has an intranet. As a result of this, all components of the CD pipeline can only be reached over this intranet. Jenkins is configured with scripts which are only one-time executed when the container is started. If Jenkins is updated then the container is deleted with all its configurations and a new one is started. The results of each CD pipeline stage are stored in a key-value database which is provided by HashiCorp Consul.

**Commit stage and repository**

The first step of the CD pipeline is that the developer commits his changes to the GitLab repository (see step 1. Figure 7.2). Jenkins triggers a new instance of this CD pipeline if changes have been committed (see step 2. Figure 7.2).

**Build stage and CI server**

If Jenkins triggers a new instance of the CD pipeline then the application is built (see step 3. Figure 7.2). Two Jenkins nodes are available for building the artifact. The build stage ends successfully if all tests have been passed. This includes the unit tests, the bean tests[1] (similar to unit tests), Checkstyle, PMD, and SonarQube.

**Artifact repository**

If the build stage is successfully run then the built artifacts are uploaded and stored in the Sonatype Nexus Repository OSS (see step 4. Figure 7.2).

**Testing stage**

If the artifact is stored successfully then the testing stage is triggered (see step 5. Figure 7.2). For testing the application has to be installed on an IBM WebSphere application server. The REST API tests are performed against this installed application.

**Production stage**

The production stage is triggered manually. If this is done the artifacts which are stored in the Sonatype Nexus Repository OSS are deployed on an SFTP server (see step 6. Figure 7.2). After that the customer can download, install and use the deployed artifact.

[1]https://github.com/NovaTecConsulting/BeanTest

### 7.1.4. Research questions

Research questions are set up to present what is expected and discovered during the case study. In this case study the following research questions are investigated:

$RQ_1$ How do team members assess the security level of the CD pipelines?

$RQ_2$ Which security objectives are implemented in the CD pipelines?

$RQ_3$ Can the selected tools be applied to the CD pipelines?

$RQ_4$ Which overall risk severities do the industrial CD pipelines have?

### 7.1.5. Methods of data collection

A list is created to detect the vulnerabilities that were determined in Chapter 5. The list of questions which is applied on **each component** of the CD pipeline includes the following questions:

1. Are access rights sufficiently restricted?
2. Is an HTTPS connection or an ssh transfer used?
3. Are vulnerable versions of components in use?
4. Can the component be shut down when someone gets access to the environment (e.g., server)?

For several CD pipeline components and stages, specific questions have to be asked in order to find their vulnerabilities. In the **commit stage** and in the **repository** the following questions are asked:

1. Does a review of code changes take place?
2. Is the CD pipeline script tested?
3. Do commit messages include sensitive data?

On the **CI server** it should be ensured that no sensitive data are shown on the UI in plain text format. The questions for the CI server are:

1. Are sensitive data shown on the CI server UI?
2. Can the configuration settings of the CI server be changed without notification?

If the **build stage** and the external **library store** is checked then the following question should be answered to detect further vulnerabilities.

1. Are the libraries checked before they are downloaded?

In the **artifact repository**, it should be considered if the artifacts stored there for deployment have vulnerabilities. That is important because the production stage is part of the CD pipeline and if a produced artifact has vulnerabilities then an attacker can harm the customer. The question to ask is as follows:

1. Have the artifacts vulnerabilities?

This questionnaire is primarily answered with the results of selected tools from Chapter 6. The preference is given to those tools which can be easily installed and integrated and which deliver understandable results. Because each CD pipeline is built differently, the tools are not always used on both CD pipelines. The tool selection criteria are that a easy installation of the tools exists and the tools provide a additional value for the company. Furthermore, there should not be the need for the knowledge of a security expert to use and understand the functionality of the tools. The tools should be actual, executable and maintained. It is found out that the detected vulnerabilities of Chapter 5 can occur in the two CD pipelines A and B.

### 7.1.6. Methods of data analysis

After collecting the vulnerabilities the results are assessed with the categorization of Section 2.5.6. Therefore, each vulnerability gets a likelihood and impact level. Out of both levels the overall risk severity is calculated. If a CD pipeline component has several vulnerabilities then the highest occurring risk level determines the security level of the CD pipeline component or stage. The CD pipeline is as secure or strong as the weakest component of the CD pipeline.

## 7.2. Case study results of CD pipeline A

This section shows the results of the investigation of the CD pipeline A. In the first part, it is shown how the project team members who use the CD pipeline assess its security level. The second subsection presents the security objectives for this specific CD pipeline. In the next step the vulnerability analysis and the detected gained results are presented. At the end of this section a summary is given on the results which are gained.

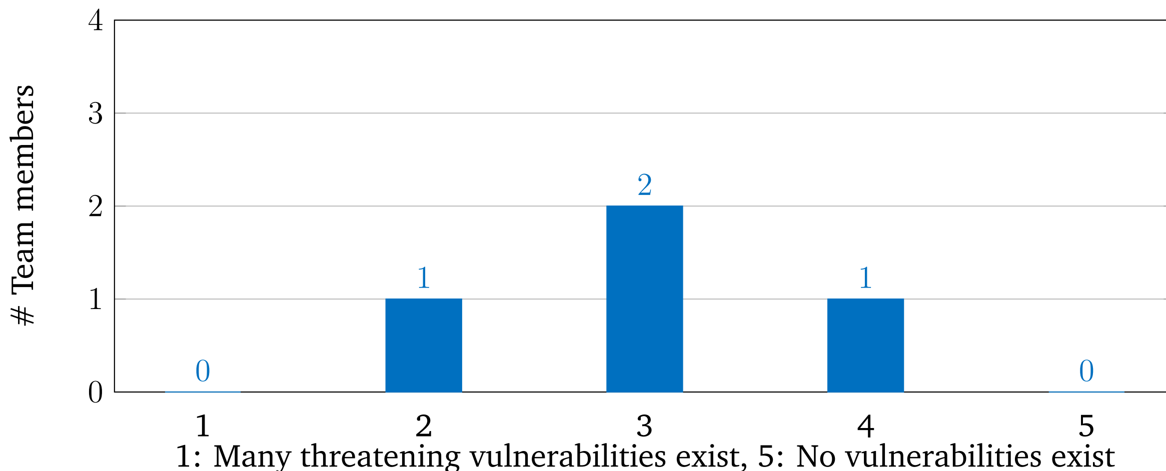



**Figure 7.3.:** Pipeline A: Security assessment through team members

### 7.2.1. Assessment of the security level of CD pipeline A by team members

The team members of the project which uses the CD pipeline A are asked in the survey (see Chapter 4) as how secure they estimate their CD pipeline. The answers are presented in the Figure 7.3. The result shows that in their views the CD pipeline is not free of vulnerabilities. On the one hand one employee thinks that the CD pipeline has threatening vulnerabilities which pose a high risk for the company (security level 2). On the other hand one employee thinks that the CD pipeline has almost no vulnerabilities (security level 4). Two employees mention that they think that the CD pipeline has vulnerabilities but not as many (security level 3).

To sum up the assessment of the employees of the CD pipeline A, they think in average that the CD pipeline has a security level of 3.

### 7.2.2. Security objectives for CD pipeline A

The project team which uses the CD pipeline A declares security objectives to guarantee a secure CD pipeline. These objectives are listed in the following.

- Perform regular updates to keep components and plugins of the CD pipeline up to date
- Restricted access and authentication is available (in most cases the project team members have administrator permissions - DevOps approach)
- Jenkins and other relevant nodes are not accessible from the public Internet (company intranet)

- Connections are encrypted (e.g., HTTPS)

It should be noted that the project team is anxious to secure their CD pipeline. Importance is attached to the security of the connection and as a result the data transfer is encrypted (e.g., HTTPS). A further step is that components are available only through an intranet connection or the components are stored in the customer's cloud foundry where it is improbable that an unauthorized person gets access to it. The project team's aim is that unauthorized persons cannot gain access to the components of the CD pipeline. Because of that, a restricted access and authentication mechanism are implemented. The project team follows the DevOps approach and as a result of this in most cases, all project team members have administrator permissions for the CD pipeline components.

If the vulnerability detection is done then it should be checked if those mentioned security objectives are observed.

### 7.2.3. Vulnerabilities detection and results

In the following the components and stages of the CD pipeline A are investigated for vulnerabilities. First, all components are investigated. Then further stages of the CD pipeline are considered. The detected vulnerabilities are presented and possible mitigation or elimination steps are shown. In the last paragraph it is decided if the security objectives are fulfilled.

#### Concerning all CD pipeline components

A manual search is performed in the VulDB for each CD pipeline A component in order to identify the vulnerabilities which it contains. For each vulnerability VulDB publishes an overall risk level. Therefore, the risk level is simply adopted from the VulDB. It was easier to execute the manual search than if the Circl CVE API were used. The result of this research is presented in Table 7.1. There it can be seen that the used Bitbucket and Jenkins versions include vulnerabilities. The detected seven vulnerabilities in Bitbucket give the component the overall risk level medium. In Jenkins vulnerabilities are only detected which have the overall risk level low. Therefore, Jenkins gets the overall risk level low. No vulnerability is known for the other components of this CD pipeline up to this point in time.

To sum up no tool was able to detect these vulnerabilities as easily as a manual inspection of the CD pipeline components. The entire CD pipeline components get the overall risk level **MEDIUM** because the weakest component, which is Bitbucket, is ranked to this level.

| Tool | Version | VulDB entries | Mitigation |
|---|---|---|---|
| Bitbucket Server | 4.14.2 | | 5.7.2 |
| CVE-2017-18088 | | LOW | |
| CVE-2017-18087 | | MEDIUM | |
| CVE-2017-18038 | | MEDIUM | |
| CVE-2017-18037 | | MEDIUM | |
| CVE-2017-18036 | | MEDIUM | |
| CVE-2017-16857 | | MEDIUM | |
| CVE-2016-4320 | | MEDIUM | |
| Jenkins | 2.89.3 | | 2.98.4 |
| CVE-2018-1000068 | | LOW | |
| CVE-2018-1000067 | | LOW | |
| JFrog Artifactory Professional | 5.8.3 | No entries | — |
| Rundeck | 2.10.2-1 | No entries | — |
| HockeyApp | — | No entries | — |

**Table 7.1.:** Pipeline A: Vulnerabilities in CD pipeline components

### Commit stage

Every team member could be able to commit malicious changes on all CD pipeline relevant scripts (Jenkinsfile, deployment scripts and Dockerfiles). For this execution the knowledge of the system and commit rights are needed. Therefore the likelihood factor is low. If code changes cannot be delivered it could have a negative impact on the companies image. The impact factor ranges between low and potentially high. As a result of this the overall risk severity is between none and medium. The project team tries to detect and eliminate this problem by reviewing each code commit. Therefore the overall risk severity is ranked to none.

Another vulnerability is that a team member can commit a CD pipeline script which is not executable. For this the knowledge of the system and commit rights are needed (likelihood factor low). If a person can commit a non-executable CD pipeline script then the damage on the company can occur. So the impact of this can be between low and potentially high. Out of these two factors the overall risk severity between none and medium is calculated. This vulnerability can be detected through the tool Jenkins Pipeline Unit. A test can be written and executed which detects these vulnerabilities before committing or during the building of the code.

Another vulnerability can be that someone commits sensitive data (e.g., credentials) through the commit message. This vulnerability can be detected through code review or the tool truffleHog. Because of the fact that code review is done before committing

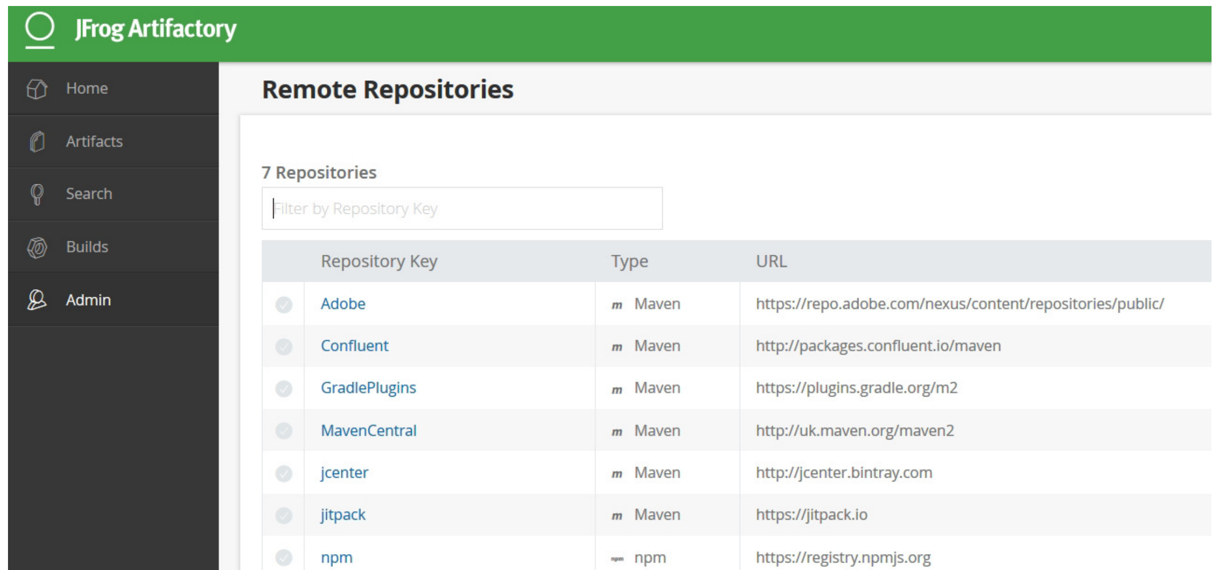



**Figure 7.4.:** Pipeline A: Unencrypted remote repository connections

it is improbable that this will happen. Therefore this vulnerability gets the overall risk severity low.

To sum up depending on the extent of the damage, the risk in the commit stage at the moment is between **LOW** and **MEDIUM**.

### Build stage and CI server

Through a manual inspection it is observed that the remote repository connections are unencrypted. Figure 7.4 shows the connections to remote repositories. There it can be seen that three of the six connections are unencrypted. As an example, if the build stage is triggered and the dependent library is not cached then it has to be downloaded from Maven Central. If the connection is using HTTP then an attacker can read the transfer traffic and manipulate or exchange the downloaded library (cross build injection attack). A proof of concept on how this can be done is given with the detection tool OWASP Zap Proxy in Section 7.4. For this attack, no access rights for the CI server are needed but the OWASP Proxy has to be installed between the Browser and the CI server. Therefore the likelihood factor is medium. It is possible that malicious code is included, which means that the impact factor is medium or potentially high. Out of these both factors the overall risk severity is between medium and potentially high. The vulnerability can be easily eliminated if an HTTPS connection is used.

In addition, the checksum of the downloaded libraries should be checked. It would be meaningful to secure each artifact of the project with a checksum. These functionalities are included in JFrog Artifctory. The tool Gradle Witness can be also

used to guarantee that the downloaded source of the libraries is correct. Because no vulnerability exists in this context no risk is present. The overall risk severity is none.

In conclusion, nearly all steps to detect the vulnerabilities in the build stage are manually performed . The complete overall risk severity of the build stage is medium or potentially high because it is possible to include malicious code in the build stage.

On the CI server UI no sensitive data is stored in plain text format. This vulnerability is prevented through the credential feature of Jenkins. There is no risk for the CD pipeline in this step and therefore the overall risk severity is none (likelihood and impact factor are low).

If an attacker gets the administrator rights of the Jenkins server then he can change the security configuration and CD pipeline jobs without any notifications. In addition to that, he can create a new CD pipeline which can delete sensitive data. The likelihood factor is low because the person needs to have full resource access rights to do this action. The impact factor can be between medium and potentially high because since this action the CI server can be opened to all unauthorized persons who know the CI server URL. Out of these two factors the overall risk severity is between low and medium. To detect and mitigate this vulnerability the plugins of Jenkins JobConfigHistory or Jenkins matrix-based security can be used to show the changes or limit the privileges of the user such that he cannot abuse these vulnerabilities.

Twistlock is one possible tool which can detect vulnerabilities in Docker images in the AWS registry. There was no possibility to test the commercial tool Twistlock.

All in all, the overall risk severity for the build stage and CI server ranges between **MEDIUM** and **POTENTIALLY HIGH**.

### Artifact repository

If an attacker can gain access to the environment where the artifact repository is installed then he can easily stop the Docker container. It is improbable that an unauthorized person is able to gain access to the cloud foundry of the customer and can damage the company. As a result of this the overall risk severity is low.

The production stage is a part of the CD pipeline. If an vulnerable artifact is installed and an attacker exploits the included vulnerability then it can harm the company's and the customer's image. Therefore it is necessary to make sure that built artifacts have no vulnerabilities. The vulnerabilities cannot be detected with JFrog Xray - only extension tool of JFrog Artifactory - because the feature to analyze gradle packages will not be released until April 2018. But to know the status about the newest artifact the OWASP dependency check is executed. The result shows that 11 of the 291 dependencies are vulnerable. The severity of the detected vulnerabilities ranges between medium and high. A brief extract of the dependency check result is located in Figure C.1. The attacker only needs access to the installed instance of the application and with an attack

he can damage the company and the customer. Because of that, the likelihood factor is low and the impact factor can be between low and potentially high.

All in all, the overall risk severity is between low and potentially high. To sum up, the vulnerable artifacts are a product of the CD pipeline but the production stage is also a part of the CD pipeline. Therefore, it is necessary to pay attention that no vulnerable libraries are used in the artifacts. The overall risk severity of the artifact repository ranges between **LOW** and **POTENTIALLY HIGH** because it includes artifacts which have vulnerable dependencies.

#### Check fulfillment of the security objectives

The first security objective of this project team is to perform regular updates to keep the components and plugins up to date. For the Jenkins component it is fulfilled because they try to update this component in regular steps. Jenkins publishes several updates per week and as a result of this it is impossible to immediately incorporate all updates. For the Bitbucket server it cannot be said that the security objectives are fulfilled. They use an old version which has vulnerabilities. The Bitbucket server is maintained by a third-party, so the fulfillment of the security objective is not in the hands of the development team.

Every component has an authentication mechanism. In all components, except for Jenkins, not all team members have admin rights. Persons only can gain access to the components if they have access rights. This security objective is fulfilled because it is also obeyed that the Jenkins master can only be accessed over the company's intranet.

The last security objective that all connections are HTTPS is not completely fulfilled because it was found out that the connections to the remote repositories are partially insecure.

In conclusion it can be said that the project team tries to obey the security objectives but they do not completely achieve them. In my opinion it is because they have little time for it and it depends on the customer and his provided infrastructure how often and what they can update.

### 7.2.4. Summary of results of CD pipeline A

To sum up the results of the detected vulnerabilities in the CD pipeline A it can be seen that there exist 11 vulnerabilities in the CD pipeline which can have an overall risk severity between **MEDIUM** and **POTENTIALLY HIGH**. A summarized overview of all detected vulnerabilities is presented in Table 7.2. It can be seen that two vulnerabilities can have the overall risk severity of potentially high. Several vulnerabilities only show an overall risk severity which is none or low. It can be seen that the employees

| Overall risk severity | | | | Number of vulnerabilities |
|---|---|---|---|---|
| NONE | | | | 3 |
| LOW | | | | 3 |
| MEDIUM | | | | 1 |
| Between | NONE | and | MEDIUM | 1 |
| Between | LOW | and | MEDIUM | 1 |
| Between | LOW | and | POTENTIALLY HIGH | 1 |
| Between | MEDIUM | and | POTENTIALLY HIGH | 1 |

**Table 7.2.:** Pipeline A: Summary of the detected overall risk severities

have accurately assessed the security level of the CD pipeline. Every component has access restrictions which means, that it is difficult for an attacker to gain access to those components. The company trusts in the cloud foundry and infrastructure of the customer and the third-party provider AWS. They have to guarantee that their part is secure. It has to be mentioned that an unaware or negligent action of project team members can lead to the exploitation of one of the vulnerabilities. Therefore, it can be helpful to mitigate or rather eliminate the detected vulnerabilities by the proposed tools.

## 7.3. Case study results of CD pipeline B

In the following the components and stages of the CD pipeline B are investigated for vulnerabilities. First, all components are investigated. Then further stages of the CD pipeline are considered. The detected vulnerabilities are presented and possible mitigation or elimination steps are shown. In the last paragraph it is decided if the security objectives are fulfilled.

### 7.3.1. Assessment of the security level of CD pipeline B by team members

The project team members which use the CD pipeline B are also asked to assess the security level of their CD pipeline. Their answers are presented in Figure 7.5. It can be seen that four team members think that there exist only some vulnerabilities on their CD pipeline which would not have high risks for the company and the customer (security level 4). Six team members think that the CD pipeline has at least a few vulnerabilities (security level 3).

To sum up the employees of the CD pipeline B think in average that the CD pipeline has a security level of 3.4.

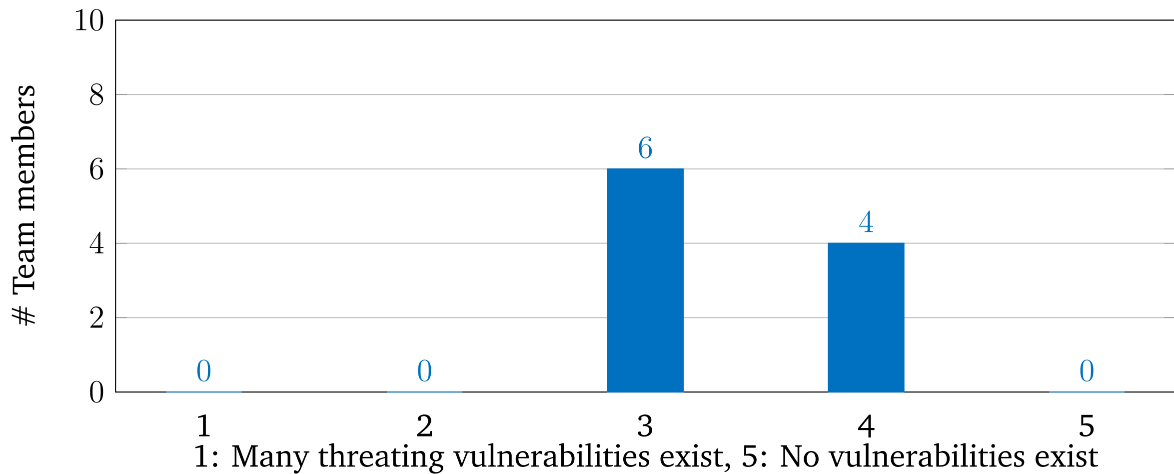



**Figure 7.5.:** Pipeline B: Security assessment through team members

## 7.3.2. Security objectives for CD pipeline B

The project team who uses the CD pipeline B has no explicit security objectives. Their main objectives are the automation and a fast velocity of the delivery process. However indirect objectives, that have a positive effect on the security of the CD pipeline, have been implemented. These are listed in the following:

- Perform regular updates to keep components and plugins of the CD pipeline up to date
- CD pipeline components are accessed only over the customer's intranet
- Restricted access and authentication mechanism, not all employees have administrator rights

The project team trusts in the intranet of the customer. Therefore, all CD pipeline components are only accessible by this means. They have the aim to update their components regularly but it is not always possible because the customer provides old virtual machines on which the CD pipeline should be implemented.

If the vulnerabilities detection is done then it should be checked if those mentioned security objectives are observed.

## 7.3.3. Vulnerabilities detection and results

The analysis proceeds in the same manner as CD pipeline A. First all components of the CD pipeline are investigated. After that vulnerabilities are detected in further stages of

CD pipeline B. For each detected vulnerability possible mitigation and elimination steps are presented. In the last part it is checked if the security objectives are fulfilled.

### Concerning all CD pipeline components

The detection of the vulnerabilities is manually done as in the CD pipeline A. In the VulDB for each component the vulnerabilities are searched and the overall risk severity is determined. The results of this research are presented in Table 7.3. It can be seen that the used Jenkins version has many vulnerabilities. 18 vulnerabilities are detected in the version of Jenkins. The Jenkins component gets the overall risk level medium because there is at least one detected vulnerability with this level available. In addition to Jenkins, the used Sonatype Nexus Repository OSS has vulnerabilities, too. Through the research three vulnerabilities are detected. Out of these three vulnerabilities one has the risk level medium. As a result of this, the artifact repository has the overall risk severity level medium. The research has discovered six vulnerabilities in the OpenSSH protocol which are used by the SFTP server for data transfer. Due to the fact that at least one of the vulnerabilities is ranked to the risk level medium, the SFTP server gets the overall risk severity level medium. For the other included components no entry in the VulDB can be found. If the detected vulnerable versions are upgraded then the vulnerabilities can be eliminated. All in all, the whole CD pipeline components are getting the overall risk severity level medium because the weakest component is Jenkins.

It should be noted that all connections between components are HTTP except the commits which are transmitted, the connection to the Jenkins nodes and the connection to the SFTP server are via SSH. If it is possible that an attacker as MITM can read and change the transferred data then the CD pipeline process can be manipulated and malicious code can be injected. To gain this information with a tool, OWSAP Zap Proxy could be appropriated. For this attack no access rights are needed. The attacker has to know the environment where the CD pipeline components are installed. Therefore, the likelihood level is ranked as medium. The impact level can range between low and potentially high because it depends on the attacker what he exploits and which damage is caused to the company. The overall risk severity of this is between low and potentially high.

The credentials of the SFTP server are known by team members but if an artifact is uploaded to the server then the customer is notified via e-mail. The released artifacts on the server can be created but cannot be overwritten or deleted. As a result of this, the overall risk severity level is none.

In conclusion, some of the components of the CD pipeline have vulnerabilities thus the whole CD pipeline components get the overall risk severity between **MEDIUM** and **POTENTIALLY HIGH**.

| Tool | Version | VulDB entries | Mitigation |
|---|---|---|---|
| GitLab Community | 10.1.0 | No entries | — |
| Jenkins | 2.32.2 | | 2.89.4 |
| CVE-2018-1000068 | | LOW | |
| CVE-2018-1000067 | | LOW | |
| CVE-2017-1000356 | | MEDIUM | |
| CVE-2017-1000355 | | LOW | |
| CVE-2017-1000354 | | MEDIUM | |
| CVE-2017-1000353 | | MEDIUM | |
| CVE-2017-1000401 | | LOW | |
| CVE-2017-1000400 | | LOW | |
| CVE-2017-1000399 | | LOW | |
| CVE-2017-1000398 | | LOW | |
| CVE-2017-1000396 | | MEDIUM | |
| CVE-2017-1000395 | | LOW | |
| CVE-2017-1000394 | | LOW | |
| CVE-2017-1000393 | | MEDIUM | |
| CVE-2017-1000392 | | LOW | |
| CVE-2017-1000391 | | MEDIUM | |
| CVE-2017-1000504 | | MEDIUM | |
| CVE-2017-1000503 | | MEDIUM | |
| CVE-2017-173832 | | LOW | |
| Nexus Repository OSS | 2.14.3 | | 3.8 |
| CVE-2018-5307 | | LOW | |
| CVE-2018-5306 | | LOW | |
| CVE-2017-17717 | | MEDIUM | |
| Hashicorp Consul | 0.7.5 | No entries | — |
| WebSphere | 8.0.0.10 | No entries | — |
| OpenSSH | 6.6.1 | | 7.7 |
| CVE-2016-10708 | | MEDIUM | |
| CVE-2017-15906 | | LOW | |
| CVE-2016-10012 | | MEDIUM | |
| CVE-2016-10011 | | MEDIUM | |
| CVE-2016-10010 | | MEDIUM | |
| CVE-2016-10009 | | MEDIUM | |
| CVE-2016-6515 | | LOW | |

**Table 7.3.:** Pipeline B: Vulnerabilities in CD pipeline component

### Commit stage

In the team of CD pipeline B every team member can commit changes on all CD pipeline relevant files (Jenkinsfile and deployment scripts). To exploit this vulnerability commit rights are required. Therefore, the likelihood factor is low. If the company cannot deliver code changes it might damage the image of the company. As a result of this, the impact factor lies between low and potentially high which means that the overall risk severity is between none and medium. The project team tries to eliminate the vulnerability through code reviews At this point of time the overall risk severity is none.

Another vulnerability is that a team member can commit a CD pipeline Groovy script which is not executable. As a result of this it is possible that no code changes can be delivered. For committing a code change access rights are needed. Therefore, the likelihood factor is low. The impact factor can be between low and potentially high because if an attacker has the ability to commit a non-executable script then code changes cannot be delivered. This results in an overall risk severity level which ranges between none and medium. The tool Jenkins Pipeline Unit can be used to detect this vulnerability. Before committing a change to a Jenkinsfile, a test is performed to ensure that this script is executable.

Another vulnerability is that someone commits sensitive data in the commit message. The exploitation of this vulnerability is improbable because a code review is done for each commit. Because of this it gets the overall risk severity level low. This vulnerability can be also detected by the tool truffleHog.

Vulnerabilities in the used Docker images cannot be detected because Docker Hub can only detect vulnerabilities in the latest official images. The installation of the other aforementioned Docker images detection tools was not possible because the tools are commercial or hard to install.

To sum up, the overall risk severity of the commit stages range between **NONE** and **MEDIUM**. This is dependent on how the attacker can damage the company.

### Build stage and CI server

All connections to the remote repositories are HTTPS connections. So the vulnerability that the downloaded libraries can be manipulated does not apply here. The Sonatype Nexus Repository OSS has included a checksum check which detects if the source of the libraries is incorrect. The tool Gradle Witness can be also used to check if the checksums are correct.

The context of the DevOps approach all team members get administrator rights of the CI server. If an attacker acquires the administrator rights of the CI server Jenkins then he is able to change the security configuration and CD pipeline jobs without any notifications. In addition to that, he can create a new CD pipeline which can delete

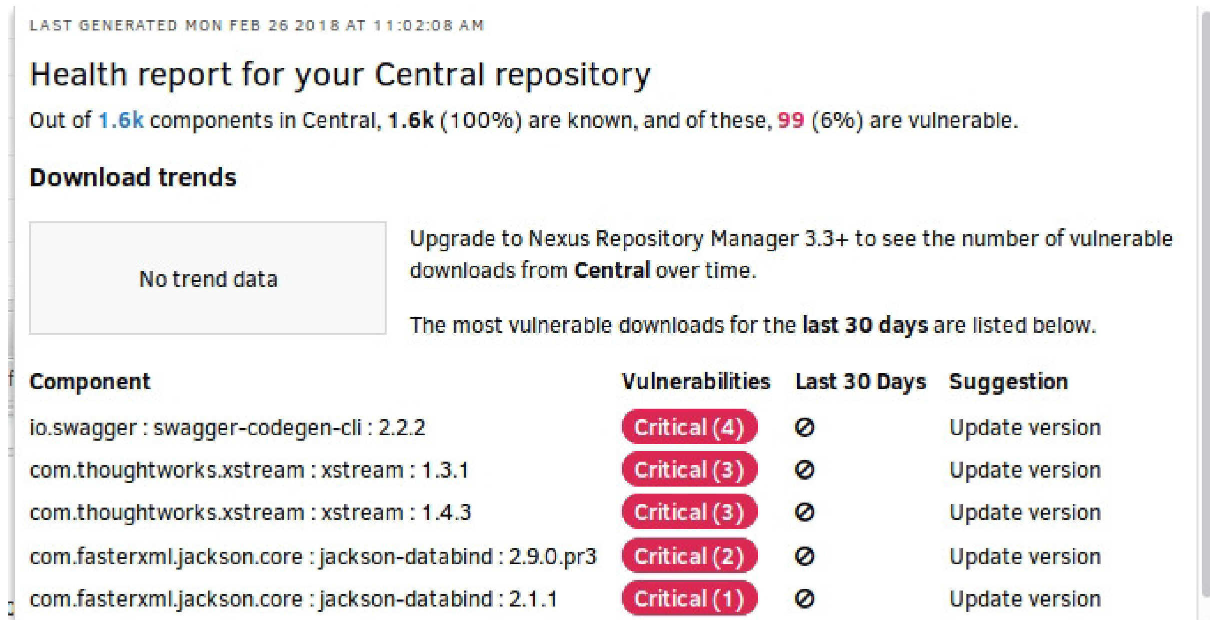



**Figure 7.6.:** Pipeline B: Results of Nexus Repository Health Check OSS

sensitive data. To do this action the full access rights of the CI server UI are needed. As a result of this, the likelihood factor is low. But the impact of this action can be that the CI server UI is open to unauthorized persons who know the CI server URL. Therefore, the impact factor ranges between medium and potentially high because sensitive data can be shown to unauthorized persons. To detect and mitigate this vulnerability the plugins of Jenkins JobConfigHistory or Jenkins matrix-based security can be used to show the changes or limit the privileges of the user such that he cannot make these changes. This means that the overall risk severity for this vulnerability ranges between **LOW** and **MEDIUM** which is also the level of the build stage and the CI server.

### Artifact repository

A vulnerability of the artifact repository is that it is possible to redeploy the artifact which means that during the step another artifact can be included. Furthermore, all artifacts are stored on the hard disk so the attacker can remove the artifacts if he gets access to the server environment. Therefore, the attacker needs full resource access rights and high system knowledge. This fact leads to a low likelihood factor. The impact factor can range between low and potentially high because it depends on the damage an attacker causes. As a result of this the overall risk severity is between none and medium. The production stage is part of the CD pipeline, so it is necessary to know if the produced artifacts are free of vulnerabilities. The integrated Sontatype Nexus Repository OSS

Health Check shows that 6% of the 1600 components are vulnerable which means 99 components. In Figure 7.6 the top six libraries with the highest security risk level are shown. It can be seen that all vulnerabilities are critical. A detailed report is not possible because the Sonatype Nexus Repository OSS 2.0 is used. If newer versions or the pro version is used then a developer can get a detailed report. These vulnerabilities can be exploited if the artifact is in production and installed. The overall risk severity, therefore, lies between medium and potentially high, because an attacker needs access to the customer's installed artifact to exploit the critical vulnerabilities. The vulnerabilities can be mitigated if the library versions are upgraded. As a result of this the artifact repository gets the overall risk severity between **MEDIUM** and **POTENTIALLY HIGH**. If the artifacts are deployed then the customer can be harmed.

**Check fulfillment of the security objectives**

The project team of CD pipeline B has no specified security objectives. From their point of view some objectives are fulfilled. These objectives are checked in the following. It is correct that all components are only accessible through the intranet of the customer. Every component of the CD pipeline is protected through an authentication mechanism. All team members have administrator rights for the Jenkins UI, Nexus UI, and Consul UI. One security objective is not fulfilled because the components are not all up to date. This is due to the fact that at the moment they cannot update the Jenkins version because they use a plugin which is not compatible with the newer Jenkins versions.

### 7.3.4. Summary of results of CD pipeline B

In conclusion it is to say, that it is easy to damage the CD pipeline if an attacker gains access to the components and their environments. The structure of the CD pipeline is reason why the CD pipeline components can be easily set up. For an attacker it is difficult to manipulate the CD pipeline if he has no access rights. It is still possible if the connections are unencrypted. The project team trusts in the company's intranet and because of the simplicity they use HTTP connections which represent the greatest threat in my opinion. To sum up the results of the detected vulnerabilities in CD pipeline B it can be seen that there exist 11 vulnerabilities in the CD pipeline which can have an overall risk severity level between **MEDIUM** and **POTENTIALLY HIGH**. A summarized overview of all detected vulnerabilities is presented in Table 7.4. It can be seen that two vulnerabilities can have the overall risk severity of potentially high. Several vulnerabilities only show an overall risk severity which is none or low.

This perception differs from that of the employees who have estimated that the CD pipeline is less vulnerable. All in all, unaware or negligent actions of project team

| Overall risk severity | | | | Number of vulnerabilities |
|---|---|---|---|---|
| NONE | | | | 2 |
| LOW | | | | 1 |
| MEDIUM | | | | 3 |
| Between | NONE | and | MEDIUM | 2 |
| Between | LOW | and | MEDIUM | 1 |
| Between | LOW | and | POTENTIALLY HIGH | 1 |
| Between | MEDIUM | and | POTENTIALLY HIGH | 1 |

**Table 7.4.:** Pipeline B: Summary of the detected overall risk severities

members can lead to the exploitation of a vulnerability. Therefore, it can be helpful to mitigate or rather eliminate the detected vulnerabilities by the proposed tools.

## 7.4. Proof of concept

Because not all tools can be tested on the CD pipelines A and B it is shown in a proof of concept how one tool can detect unencrypted connections.

**Proof of concept: Insecure connection**

If there exist unencrypted connections on account of using HTTP then it is possible for an attacker to read and manipulate the transferred data. The following shows that an attacker has the ability to exchange the downloaded libraries if the connection to the remote repository Maven Central is HTTP. The requirements for the attack is a simple gradle project which has included in the build.gradle file (see Listing C.2) an HTTP connection to Maven Central. Additionally, in the gradle.properties file the proxy host and port have to be set (see Listing C.1). After that the OWASP Zap Proxy has to be installed and the same proxy configurations have to be set. If the project is built with the command *gradle build* OWASP Zap Proxy records which libraries were downloaded. The record output is shown in Figure 7.7. For example it can be seen that the joda-time library 2.2 is downloaded. Through OWASP Zap Proxy the attacker is able to exchange the library.

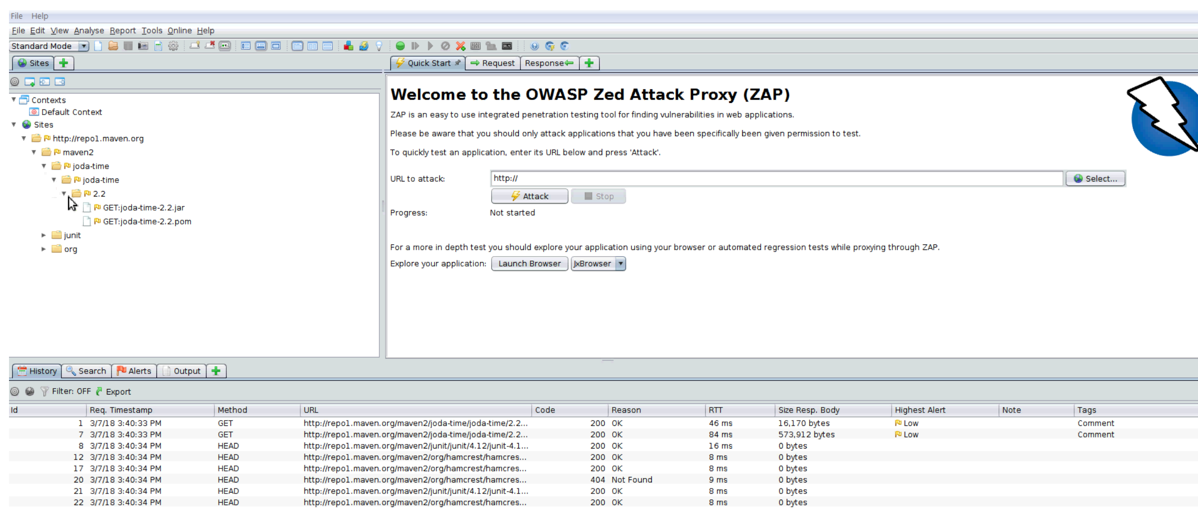



**Figure 7.7.:** Detection of vulnerabilities with OWSAP Zap Proxy

## 7.5. Discussion of both investigated CD pipeline results

During both investigations of the CD pipelines it should be found out how both project teams assess the actual security level of their CD pipeline. The project team A assess a security level of 3 and the project team B a security level of 3.4.

In the second step of these investigations it should be found out if there exist security objectives defined in both CD pipelines and if they are fulfilled during the execution and operation of the CD pipeline. The results show that only the project team A has declared explicit security objectives for their CD pipeline. The project team B pursues indirectly security objectives but their main aims are the automation and fast velocity of the delivery process. All in all, both teams try to fulfill their objectives but project team A does this with more awareness. Both teams cannot fulfill the security objectives completely because they are dependent on the network, hardware and cloud infrastructure of the customer. Often they would like to do more in this area but they have no budget and time for it or components are managed by a third-party.

There are tools which are useful to detect vulnerabilities in CD pipelines. They require little effort to be integrated and need no security expert knowledge for usability and interpretation of their results. A few detections were performed manually because it was easier and faster to do this on the CD pipelines A and B. Moreover, a half of the tools can prevent certain vulnerabilities in these two CD pipelines. They help to mitigate and eliminate the vulnerabilities in advance. To sum up there exist tools which can increase the security level of the CD pipelines A and B.

The investigation of the CD pipelines shows that there exist in both CD pipelines vulnerabilities which can have an overall risk severity of potentially high. Through

the fact, since the CD pipeline components are nearly all installed on the customer's infrastructure or on third-party provider the two project teams have to trust that the customers and the third-party providers secure their network and hardware. The project team B has assessed the security level of their CD pipeline worse than the project team A. In my opinion, the highest risk at the moment is an unaware action of a project team member, an employee of the customer or an attack on the third-party which would be able to exploit the existing vulnerabilities.

Chapter 8

# Conclusion

In this chapter a summarized overview of the thesis research is given. The first part presents the feedback of the project teams of both investigated CD pipelines A and B (Section 8.1). This also includes their reactions to the detected vulnerabilities. The chapter further discusses the applicability of the research results to industrial CD pipelines from a general perspective (Section 8.2). In the next part, the research questions from the beginning are taken up and the results are summarized for each of them (Section 8.3). The last part of this chapter contains some lessons learned (Section 8.4) and future works (Section 8.5).

## 8.1. Feedback of the project teams on the results

This section provides feedback from both project teams on the detected vulnerabilities. It also shows what further steps they will take to secure their CD pipeline. The detected vulnerabilities were also verified by the project teams as such.

**CD pipeline A**
The reaction of the project team on the detected vulnerabilities of CD pipeline A was that they immediately changed the remote repository connection from HTTP to HTTPS. They were surprised that they use such an old version of Bitbucket but they want to make this update at a later time. In addition, they will try to mitigate all possible detected vulnerabilities because they know that these improvements can increase the security level of their CD pipeline.

**CD pipeline B**
The project team responded very positively to the discoveries. They mentioned that it

did happen once that someone pushed a non-executable CD pipeline script. They saw the Jenkins Pipeline Unit as a good idea because the integrated analyzes can be very useful. Furthermore, they saw it as unrealistic that someone will commit sensitive data in the commit messages. They agreed that they should use the actual dependency library without a vulnerability. The mitigation of these vulnerabilities are difficult because they have the problem of using the programming language Java version 6, which some newer versions of libraries do not support. They will try to improve these kinds of vulnerabilities in the future. In addition, they thought that it is unrealistic that they would change all HTTP connections in HTTPS because they would need to use self-signed certificates to do this. They were not sure if they can get these self-signed certificates from the customer and then they have to trust this customer root. Maybe there will be a less complex solution which they can use to switch from HTTP to HTTPS, but at first, they have to find it out. This project team also wanted to try to mitigate all possible vulnerabilities because they know that by this way they could increase the security level of their CD pipeline.

## 8.2. General applicability of the research results

The results of the case study show that the detected vulnerabilities, such as unencrypted connections, can be found in both projects. The problem is that not many employees of the company deal with matters of security. As a result of this, they have little awareness about security vulnerabilities and the potential consequences of their exploitation. In addition, it was found out that not every project team has defined concrete security objectives for their CD pipelines. Often the focus lies exclusively on fast velocity and automation of the CD process and not on any security aspects. In both teams there exists a dependence on the budget, time and infrastructure of the customer. In my opinion, these aforementioned aspects are also problems of other CD pipelines in the industry. If there is no budget and no time available then project teams will give up the security of their CD pipelines.
Another observation of the case study was that every CD pipeline looks different. For each CD pipeline, a project team has to analyze which tools can be used to detect its vulnerabilities. If a project team in the industry use the Jenkins CI server, for example, then the Jenkins Unit Pipeline tool or the JobConfigHistory plugin can be applied.
The case study results show that the selected tools can prevent or timely detect vulnerabilities in CD pipelines. I think it is as true for vulnerabilities as it is for tests that if vulnerabilities are detected as early as possible, a company can save money. In the CD pipelines A and B, almost no security detection or mitigation tools are used. In my opinion, the situation in the industry is similar or worse.

According to Dijkstra [Dij72], “program testing can be a very effective way to show

the presence of bugs, but is hopelessly inadequate for showing their absence". I think that this statement holds just as true for vulnerability detection in CD pipelines. This opinion can be supported by the statement of Linus Torvalds[1] (creator of the Linux kernel) which mentions that "security problems are just bugs" [Tor17]. In industrial projects, the selected tools can be an effective way to show that vulnerabilities are present in CD pipelines, but they cannot show their absence. In my opinion, nobody can prove that a CD pipeline is free of vulnerabilities.

## 8.3. Summary

The aim of this thesis was to detect vulnerabilities in essential elements and in the structure of CD pipelines. Six research questions were addressed in this thesis.

To achieve the first research question ($RQ_1$) several detection approaches and methods were researched. One approach of this was the threat modeling approach which is used to detect vulnerabilities in CD pipelines.

The aim of the research question two ($RQ_2$) was to find out the work which was already conducted in this thesis field. There are a few works which deal with securing CD pipelines through developed tactics or tools. These works differ from this thesis.

With the third research question ($RQ_3$) it was found out what knowledge given company's developers have in the area of securing and vulnerability detection of CD pipelines. To gain this knowledge, a survey was conducted. 19 employees of the company participated in the survey. The results of this survey were that nearly all employees work with CD pipelines but on average they only occasionally interact with matters of security. In the view of the employees there exist several entry points which an attacker can use to harm the CD pipeline. An additional result was that in the view of the developers the CIA security attributes (confidentiality, integrity and availability) are the most important ones.

The aim of the fourth research question ($RQ_4$) was to find out which vulnerabilities exist with regard to CD pipelines. For the detection of vulnerabilities the STRIDE method of the threat modeling approach was executed on a generalized CD pipeline. Only those vulnerabilities which can harm the CIA security attributes were considered. It was found out that CD pipelines can have several vulnerabilities. If the connection between two CD pipeline components is unencrypted, for example, an attacker would have the ability to include malicious code into the CD pipeline.

By the fifth research question ($RQ_5$), several tools have been selected which are able to detect vulnerabilities in CD pipelines automatically. The results of this research

[1] https://www.linuxfoundation.org/projects/linux/

were that for each CD pipeline stage and component at least one tool exists which can be used to detect or mitigate the vulnerabilities in CD pipelines.

The sixth and last research question ($RQ_6$) has the aim to find out how secure specific CD pipelines in a company are. In order to answer this question, a case study was conducted. In that case study for each CD pipeline stage one of the selected tools were used to detect the vulnerabilities in the investigated CD pipelines A and B. Some vulnerabilities were manually detected because it was easier. The results of the case study show that both investigated CD pipelines have included vulnerabilities which have an overall risk severity between medium and potentially high. A few vulnerabilities in those two CD pipelines occur because the project teams are dependent on the infrastructure of the customer. In addition, it was found out that an unaware action of an employee can lead to an accidentally exploitation of this detected vulnerability. For both investigated CD pipelines the project teams have to trust in security of the foreign infrastructure and their own team members.

All in all, the research shows which vulnerabilities are present in CD pipelines and how they can be detected. The CD pipeline is as secure as its weakest component. If the customer has no understanding of security aspects then it is impossible for the company to perform tasks to improve the security level of CD pipelines. The company's project teams nonetheless try to mitigate the detected vulnerabilities. If these mitigations are done then the security level of their CD pipelines can be increased.

## 8.4. Lessons learned

During the research it was partially not possible to integrate all selected tools into the CD pipelines A and B. Often the help of the customer was needed because the project teams cannot change any properties or settings of the infrastructure of the customer. In some cases available tools are partially too old or not appropriate for use because they can detect vulnerabilities only in special CD pipeline components which are not used. In most cases, it was easier to do the detection manually before integrating a tool. Because each CD pipeline is different, each CD pipeline has to be analyzed to determine which tools can be used to detect vulnerabilities.

Before I started the research I thought that all employees who use the CD pipeline do know what can happen if security is neglected in CD pipelines and vulnerabilities are included. But the reality shows that there are only a few employees who know the cases. The others are unaware because they do not deal matters of security.

The security aspects are not established such as functional tests which are integrated at the beginning of the software development process to detect errors. In my opinion, the company managers have to dictate that they want the developer teams to pay more attention to the security aspects. If this would be an aim the customer has to be

convinced that he should provide budget, additional time and up-to-date infrastructure for these tasks. In the current situation the customer restricts the development teams in this work.

## 8.5. Future Work

Since only two industrially used CD pipelines of a selected company were investigated, further CD pipelines of other companies could be tested to identify additional vulnerabilities in existing CD pipelines. A case study with several participants from different companies could confirm that the CD pipelines used by most companies have vulnerabilities where the overall risk level is between medium and potentially high.

Since no tool was found to monitor the entire CD pipeline and automatically detect several vulnerabilities, such a cockpit tool could be developed.

In addition, some strategies could be developed to convince employees and customers to consider security aspects in CD pipelines. The aim should be that the customer and companies provide budget, time, and an up-to-date infrastructure for these activities.

Another opportunity for future work may be that CD pipelines are described by models. These models are then checked for the security attributes.

Appendix A

# Complete design and results of the survey

---

Appendix A contains design and the complete results of the survey. In Figure A.1 a detailed overview of the results of question 2 is given. The question deals with the security attributes ranking (further details on the survey see Chapter 4). Figure A.2 shows the development experience of the survey participants in years. From these results, the mean value is calculated, which is presented in Chapter 4 when presenting the results of question 1 of the survey.

# A. Complete design and results of the survey

Detection of vulnerabilities in CD pipelines — Load unfinished survey — Exit and clear survey

## Detection of vulnerabilities in CD pipelines

*Welcome to the survey "Detection of vulnerabilities in CD pipelines".*

*My name is Christina Paule and at the moment I am writing my Masterthesis about "the detection of vulnerabilities in CD pipelines".*

*The aim of my thesis is to detect the security vulnerabilities in the structure of CD pipelines. The focus is not leading on the detection of security vulnerabilities in the source code.*

*For my thesis I need your help. Please fill out my survey that I can get your opinion and knowledge about the security aspects in respect to CD pipelines.*

*The survey is short and does not need much time to fill out. The data is collected anonymously. You are always entitled to end the survey.*

*You would help me a lot if you would take the time to answer my questions.*

*Thank you in advance for your participation.*

There are 9 questions in this survey.

A note on privacy
This survey is anonymous.
The record of your survey responses does not contain any identifying information about you, unless a specific survey question explicitly asked for it. If you used an identifying token to access this survey, please rest assured that this token will not be stored together with your responses. It is managed in a separate database and will only be updated to indicate whether you did (or did not) complete this survey. There is no way of matching identification tokens with survey responses.

## Questions to vulnerabilities in respect to continuous delivery (CD) pipelines

* In your opinion which security objectives should be pursued in respect to CD pipelines? Please do not focus on a specific used pipeline. Think in general.

The answer should include the security objectives of a pipeline, not the security objectives in respect to the developed source code.

In your opinion which security attribute is the most important one in respect to CD pipelines (artifacts, files, scripts, connections, ...). Order the following security attributes according to their importance. The attribute on top is the most important one for you.

Double-click or drag-and-drop items in the left list to move them to the right - your highest ranking item should be on the top right, moving through to your lowest ranking item.

| Your choices | Your ranking |
|---|---|
| Confidentiality: ensures that only authorized persons gain access to the data. | |
| Integrity: ensures that an unauthorized person can not manipulate the data. | |
| Availability: ensures that the service is accessible when it is needed. | |
| Authorization: ensures that the specified identity matches with the user. | |
| Authentication: ensures that the user can only perform the activities for which he has rights/privileges. | |
| Non repudiation: ensures that the user can not deny what he has done. | |

* In your opinion what are possible attack scenarios for the pipeline you use? Against which attacks would you like to protect your pipeline?

* Which security objectives are pursued in your project in respect to CD pipelines? Which are implemented?

* How many years of experience in software development do you approximately have?

Only numbers may be entered in this field.

years: rounded integer

* Which tools do you know and/or use?

Check all that apply

- [ ] Jenkins
- [ ] Kubernetes
- [ ] TeamCity
- [ ] Spinnaker
- [ ] Travis
- [ ] GoCD
- [ ] Concourse CI
- [ ] PMD
- [ ] Checkstyle
- [ ] FindBugs
- [ ] FindBugs Securtiy
- [ ] JFrog Artifactory
- [ ] OWASP ZAP
- [ ] BDD Security
- [ ] JFrog Xray
- [ ] Security Monkey
- [ ] Blackduck
- [ ] Snyk

## Gerneral questions

* In which roles do you interact with your CD pipeline?

Check all that apply

- User: Committing code to the project, usage of the UI of the CD pipeline.
- Installation and operation of the pipeline.
- Configuration of the pipeline
- Other:

* In your opinion how important is the topic security vulnerabilities in CD pipelines?

1 2 3 4 5

1: not important

5: very important

* How often do you deal with security in your development process?

Choose one of the following answers

- Never
- Only occasionally
- Quite often
- Most of the time
- No answer

---

In the next step think about the security of the [redacted] CD pipline. In your opinion how secure is this pipeline?

1 2 3 4 5

1 means CD pipeline is insecure

5 means CD pipeline is secure (pipeline has no vulnerabilities)

## Results

### Survey 326611

| | |
|---|---|
| Number of records in this query: | 15 |
| Total records in survey: | 15 |
| Percentage of total: | 100.00% |

**Quick statistics**
Survey 326611 'Detection of vulnerabilities in CD pipelines'

## Field summary for F001

In your opinion which security objectives should be pursued in respect to CD pipelines? Please do not focus on a specific used pipeline. Think in general.

| Answer | Count | Percentage |
|---|---|---|
| Answer | 15 | 100.00% |
| No answer | 0 | 0.00% |
| Not displayed | 0 | 0.00% |

| ID | Response |
|---|---|
| 13 | 1) The CI pipeline should include a suite a security tests, tests that need to be written and maintained.<br>2) You must guard the guards - the CI infrastructure itself must likewise be secure. |
| 21 | authentication<br>authorization which user can read/start/edit pipelines |
| 24 | In general, a pipeline should not compromise any of the security objectives of the application it delivers. |
| 30 | Pipeline can not be modified from a third person<br>Pipeline builds are repeatable to be able to compare builds |
| 36 | Access rights, prohibiting unallowed users to trigger deployments to PROD but DEV/INT/TEST as well.<br>Securing as well source code, (build/deploy) logs and built artifacts.<br>Using personalized users to track code pushes or build/deployment triggers on a user basis (of course technical users are necessary as well to automatically trigger builds and deployments after a code commit or integration (performed by a personalized user).<br>Securing environment properties like login data for PROD databases etc. which are necessary for the deployed application but should be only visable/accessible to an authorized set of users. These access rights may differ between the different environments (every developer may access and modify the configuration for DEV but only very few people for PROD).<br>ATTENTION: The differences described above regarding the access rights to DEV and PROD environments and their properties are given with a focus on "traditional separation of software development and operations". Following the culture of DevOps, these borders disappear and all developers are responsible for the production as well. Therefore they of course need all access to it and its configuration then. |
| 40 | Securing credentials<br>Encrypted connections between the Buildserver(s)<br>User- & Rights-Management |
| 42 | - Ensure that the console/log/... output or any other artifacts accessible within the pipeline is not given to anyone. The access to these artifacts should be controlled.<br>- Do not allow a anonymous manual deployment. |
| 43 | repeatable builds, integrity |
| 47 | not everybody should have access, trace which person changes the pipeline, |
| 48 | access restriction, not storing credentials in plain text |
| 53 | - integrity: configuration/ build steps ... should not be manipulated from the outside / source code should not be manipulated -> otherwise, the customer gets a manipulated artifact |
| 64 | * Always use secure communication over SSH and/or HTTPS<br>* reduce human failure (passwords on the desk, passwords in the garbage)<br>* only keep security in the mind<br>* use the 4-eye princip<br>* communicate clear place where to store (common) passwords<br>* do not give passwords only to "1 or 2 trustworthy" persons, the other persons will find a way |
| 67 | Only authorized people should be able to trigger and manage the pipeline. It should be garantueed that the deployed artifacts are exactly build from the source code and no harmfull code can be injected. The dependencies (libraries, base images, ...) must validated to have no vulnerabilities. |
| 69 | A continuous deployment pipeline needs to have permissions to access sensitive systems like source code repositories and production environments to do its job. These systems might contain sensitive data like for example application credentials to backend systems or a production database with customer data. |

| | |
|---|---|
| | In my opinion, the main focus should be to prevent users from gaining unwanted access to the systems connected to the pipeline, either by extracting credentials from the pipeline or by misusing it in order to perform operations for which they normally would not be authorized. |
| 73 | Access rights should be valued, thus no authorized user can modify or run any pipelines. The outcome of the pipeline should be not modifiable by the pipeline-tool so that no vulnerabilities can be introduced using the tool itself (For example introducing malicious artifacts or weak passwords as pipeline parameters). The pipeline itself should check, whether the artifacts produced are secure. |

**Quick statistics**
Survey 326611 'Detection of vulnerabilities in CD pipelines'

## Field summary for F003

In your opinion what are possible attack scenarios for the pipeline you use? Against which attacks would you like to protect your pipeline?

| Answer | Count | Percentage |
|---|---|---|
| Answer | 15 | 100.00% |
| No answer | 0 | 0.00% |
| Not displayed | 0 | 0.00% |

| ID | Response |
|---|---|
| 13 | - |
| 21 | Denial of Service<br>If the CD pipeline can push a software package directly to production and the hacker can also change software in the VCS, it may be possible to execute bad code in production system. This should not be possible. |
| 24 | * any availability hazard: an unavailable pipeline would prevent us from delivering software<br>* unauthorized access: a non-malicious user can misconfigure a component of the pipeline |
| 30 | Changes on CI/CD Server configuration |
| 36 | Grapping configuration files to directly access databases or backends and perform data manipulation there.<br>Getting the source code as the intelectual property and publish and sell it under a different name.<br>Integrate "bad code" like backdoors or special behaviours (like: don't charge my account for shop orders) into the application going productive later. |
| 40 | Anybody can change/access the pipeline<br>Pipeline uses old libraries/applications with vulnerabilities |
| 42 | - try a manipulation with the input data a job requires when it is started manually<br>- zero day exploits against the server runnning the CI/CD software<br>- bruteforce use/passwords<br>- trying to run scripts to gain root access to the filesystem where the CI/CD server runs or the agent |
| 43 | Manipulate files/artifacts, unauthorized use of resources |
| 47 | Changes are being made by individuals that break the pipeline, Updates of the environment where the pipeline runs destroys the pipeline, |
| 48 | plant malicious code, gain access to build hosts, retrieve credentials |
| 53 | - source code manipulation -> bad for integrity<br>- DoS attacks -> bad for availability |
| 64 | - stolen passwords / ssh keys<br>- amok user delete all jenkins jobs<br>- |
| 67 | All of the above mentioned apply for our toolchain, I think. |
| 69 | I'll list some kinds of attacks that come to mind immediately:<br>- Malicious code alteration, either by commit directly to our version control system, or by modifying the final build artifact which is deployed to production.<br>- Extraction of credentials from the pipeline configuration in order to use them to get access to sensitive systems connected to the pipeline.<br>- Modifying the pipeline configuration to make it perform additional operations, which normally would not be permitted.<br><br>I would like to protect our pipeline against all attacks. In practice, there is always a trusted circle of developers/operators which we cannot really protect from. |
| 73 | Possible attacks scenarios are running the pipeline to release an manipulated artifact. The code to create the artifact was manipulated by the attacker beforehand. The only obstacle left to overcome to release a malicious artifact is the CD pipeline. Another secanrio is that the attacker introduces weaknesses over the pipeline. Maybe the attack has no access to the code but can introduce weak password or vulnerable dependencies over the CD pipeline parameters. |

## Field summary for F004

Which security objectives are pursued in your project in respect to CD pipelines? Which are implemented?

| Answer | Count | Percentage |
|---|---|---|
| Answer | 15 | 100.00% |
| No answer | 0 | 0.00% |
| Not displayed | 0 | 0.00% |

| ID | Response |
|---|---|
| 13 | As of now we've a suite of security tests that runs, and must pass, alongside all the other test classes. |
| 21 | Authentication and Authorization (as I know) |
| 24 | * access credentials are securely stored, can be easily changed and - where potentially accessible to a larger group, e.g. in the source repository - they are encrypted<br>* infrastructure automation that allows us to repave: we can quickly restore almost all components of the pipeline to a working configuration |
| 30 | Code checkout only with SSH key<br>Pipelines are checked in as code |
| 36 | Too less!<br>Builds and (TEST) deployments are only accessible to authorized users with RBAC on the different project's pipelines.<br>Source code is only visible to authorized developers (but ALL of them independent from their (sub-) project.<br>PROD deployments in the traditional environment can only be done by a special ops team. In the DevOps environment all responsible developers have access to PROD.<br>There is no established process for authorizing new users |
| 40 | none |
| 42 | - authentication<br>- probably authorization<br>- keep an more or less actual stack (keep the software up2date)<br>- hide critical data (user/pw/...) from the console output (I hope so!) |
| 43 | authorization, authentication |
| 47 | access control, pipeline flow in git to revert changes |
| 48 | secured credentials, access restriction to push code, authorization, authentication, file permissions |
| 53 | - Integrity/ Authorization: Only authorized users with the corresponding rights can change the source code (e.g push on the master branch)<br>- Authorization: Only authorized users with the corresponding rights can manipulate the build process in the build server (e.g Jenkins) |
| 64 | j |
| 67 | - Authorization of users -> Only authorized people can trigger and update the pipeline and only with the correct privileges<br>- Availability was not an issue, but is not taken care of so much at the moment |
| 69 | Our main objectives are that is<br>1) Shouldn't be possible for unauthorized persons to introduce (potentially malicious) code to our PROD deployment.<br>2) All security credentials which are part of the CD pipeline or the applications it delivers are protected from persons that do not need to know about them.<br><br>We implement the following measures:<br>- We have a strict review process, where at least two people have to review every code change before it can be commited to the central version control system.<br>- Credentials are only usefull in combination with firewall whitelisting, i.e. access to most sensitive systems is only possible from the pipeline servers itself, not from arbitrary systems. This protects against stolen credentials being misused.<br>- The pipeline is set up by a single small team, all other development teams do not have administrative access to it.<br>- Permissions on all systems which are parts of the pipeline are restrictive per default and then only the permissions are granted to individual developers that are necessary.<br>- No sensitive information should be included in the source code of applications that other |

| | |
|---|---|
| | teams have access to as well. |
| 73 | From the ranked list above in my project all objectives are pursued. Also all are implemented expect for availability. It may happen, that the CD pipeline is blocked or down. However in most cases this isn't caused by an attacker but by missconfiguration or maintenance. |

## Field summary for F004

Which security objectives are pursued in your project in respect to CD pipelines? Which are implemented?

| Answer | Count | Percentage |
|---|---|---|
| Answer | 4 | 100.00% |
| No answer | 0 | 0.00% |
| Not displayed | 0 | 0.00% |

| ID | Response |
|---|---|
| 3 | - Perform regular security updates for the pipeline<br>- Authentication for accessing the pipeline, the version control system and the artifact repository (Implemented)<br>- Fine grained authorization (Only partly implemented, e.g. every developer has full access to the general pipeline system)<br>- DoS protection (depends on the cloud provider, in our case AWS) |
| 4 | * jenkins and other relevant nodes are not accessible from the public internet<br>* authentication is guaranteed through passwords (password strength?)<br>* we use https |
| 6 | We keep our systems up to date. For example Jenkins updates and install the newest plugins.<br>We also have restricted access to our systems. Only special users can configure and work with our systems. |
| 7 | Authentication and Authorization on Jenkins, BitBucket, Artifactory and Build Nodes. |

**Quick statistics**
Survey 859357 'Detection of vulnerabilities in CD pipelines [redacted]

## Field summary for F004

Which security objectives are pursued in your project in respect to CD pipelines? Which are implemented?

| Answer | Count | Percentage |
|---|---|---|
| Answer | 4 | 100.00% |
| No answer | 0 | 0.00% |
| Not displayed | 0 | 0.00% |

| ID | Response |
|---|---|
| 3 | - Perform regular security updates for the pipeline<br>- Authentication for accessing the pipeline, the version control system and the artifact repository (Implemented)<br>- Fine grained authorization (Only partly implemented, e.g. every developer has full access to the general pipeline system)<br>- DoS protection (depends on the cloud provider, in our case AWS) |
| 4 | * jenkins and other relevant nodes are not accessible from the public internet<br>* authentication is guaranteed through passwords (password strength?)<br>* we use https |
| 6 | We keep our systems up to date. For example Jenkins updates and install the newest plugins.<br>We also have restricted access to our systems. Only special users can configure and work with our systems. |
| 7 | Authentication and Authorization on Jenkins, BitBucket, Artifactory and Build Nodes. |

## Field summary for F004

Which security objectives are pursued in your project in respect to CD pipelines? Which are implemented?

| Answer | Count | Percentage |
|---|---|---|
| Answer | 4 | 100.00% |
| No answer | 0 | 0.00% |
| Not displayed | 0 | 0.00% |

| ID | Response |
|---|---|
| 3 | - Perform regular security updates for the pipeline<br>- Authentication for accessing the pipeline, the version control system and the artifact repository (Implemented)<br>- Fine grained authorization (Only partly implemented, e.g. every developer has full access to the general pipeline system)<br>- DoS protection (depends on the cloud provider, in our case AWS) |
| 4 | * jenkins and other relevant nodes are not accessible from the public internet<br>* authentication is guaranteed through passwords (password strength?)<br>* we use https |
| 6 | We keep our systems up to date. For example Jenkins updates and install the newest plugins.<br>We also have restricted access to our systems. Only special users can configure and work with our systems. |
| 7 | Authentication and Authorization on Jenkins, BitBucket, Artifactory and Build Nodes. |

**Quick statistics**
Survey 859357 'Detection of vulnerabilities in CD pipelines

## Field summary for F004

Which security objectives are pursued in your project in respect to CD pipelines? Which are implemented?

| Answer | Count | Percentage |
|---|---|---|
| Answer | 4 | 100.00% |
| No answer | 0 | 0.00% |
| Not displayed | 0 | 0.00% |

| ID | Response |
|---|---|
| 3 | - Perform regular security updates for the pipeline<br>- Authentication for accessing the pipeline, the version control system and the artifact repository (Implemented)<br>- Fine grained authorization (Only partly implemented, e.g. every developer has full access to the general pipeline system)<br>- DoS protection (depends on the cloud provider, in our case AWS) |
| 4 | * jenkins and other relevant nodes are not accessible from the public internet<br>* authentication is guaranteed through passwords (password strength?)<br>* we use https |
| 6 | We keep our systems up to date. For example Jenkins updates and install the newest plugins.<br>We also have restricted access to our systems. Only special users can configure and work with our systems. |
| 7 | Authentication and Authorization on Jenkins, BitBucket, Artifactory and Build Nodes. |

| | Rank 1 | Rank 2 | Rank 3 | Rank 4 | Rank 5 | Rank 6 |
|---|---|---|---|---|---|---|
| Confidentiality | 3 | 3 | 6 | 4 | 2 | 1 |
| Integrity | 5 | 6 | 2 | 5 | 1 | 0 |
| Availability | 6 | 5 | 1 | 2 | 1 | 4 |
| Authorization | 1 | 3 | 4 | 1 | 5 | 5 |
| Authentification | 3 | 2 | 6 | 5 | 3 | 0 |
| Reputation | 1 | 0 | 0 | 2 | 7 | 9 |

**Figure A.1.:** Survey question 2 results: Assessment of the six security attributes through developers

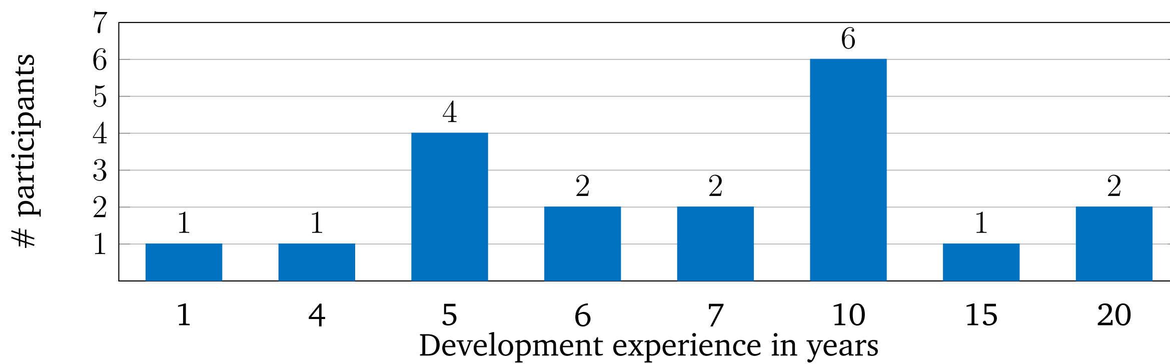



**Figure A.2.:** Survey question 1 results: Development experience of the participants

Appendix B

# Insecure Jenkins configuration XML file

---

Appendix B contains the insecure Jenkins configuration XML file which is used to open the Jenkins UI for unauthorized persons. This file is used to shut off the security properties of Jenkins. To get this configuration file, Jenkins[1] must first be installed and secondly, the security configurations have to be disabled via the Jenkins UI. The folder Jenkins contains the config.xml.

[1]https://jenkins.io/download/

**Figure B.1.:** Insecure Jenkins configuration XML

```xml
<?xml version='1.0' encoding='UTF-8'?>
<hudson>
  <disabledAdministrativeMonitors/>
  <version>2.103</version>
  <installState>
    <isSetupComplete>true</isSetupComplete>
    <name>RUNNING</name>
  </installState>
  <numExecutors>2</numExecutors>
  <mode>NORMAL</mode>
  <useSecurity>true</useSecurity>
  <authorizationStrategy class="hudson.security.AuthorizationStrategy$Unsecured"/>
  <securityRealm class="hudson.security.SecurityRealm$None"/>
  <disableRememberMe>false</disableRememberMe>
  <projectNamingStrategy
  class="jenkins.model.ProjectNamingStrategy$DefaultProjectNamingStrategy"/>
  <workspaceDir>${JENKINS_HOME}/workspace/${ITEM_FULL_NAME}</workspaceDir>
  <buildsDir>${ITEM_ROOTDIR}/builds</buildsDir>
  <markupFormatter class="hudson.markup.EscapedMarkupFormatter"/>
  <jdks/>
  <viewsTabBar class="hudson.views.DefaultViewsTabBar"/>
  <myViewsTabBar class="hudson.views.DefaultMyViewsTabBar"/>
  <clouds/>
  <scmCheckoutRetryCount>0</scmCheckoutRetryCount>
  <views>
    <hudson.model.AllView>
      <owner class="hudson" reference="../../.."/>
      <name>all</name>
      <filterExecutors>false</filterExecutors>
      <filterQueue>false</filterQueue>
      <properties class="hudson.model.View$PropertyList"/>
    </hudson.model.AllView>
  </views>
  <primaryView>all</primaryView>
  <slaveAgentPort>50000</slaveAgentPort>
  <disabledAgentProtocols>
    <string>JNLP-connect</string>
    <string>JNLP2-connect</string>
  </disabledAgentProtocols>
  <label></label>
  <crumbIssuer class="hudson.security.csrf.DefaultCrumbIssuer">
    <excludeClientIPFromCrumb>false</excludeClientIPFromCrumb>
  </crumbIssuer>
  <nodeProperties/>
  <globalNodeProperties/>
</hudson>
```

Appendix C

# Results and used file excerpts of the case study

In appendix C an excerpt of the build.gradle file (see Listing C.2) and the content of the gradle.properties file (see Listing C.1) are shown. These files are used in the proof of concept (see Section 7.4) to detect unencrypted connections with the tool OWASP Zap Proxy. The last part of this appendix includes the OWSAP dependency check results of pipeline A (see Figure C.1).

**Listing C.1** Unencrypted connection detection with OWASP Zap Proxy: gradle.properties

```
1         systemProp.http.proxyHost = 127.0.0.1
2         systemProp.http.proxyPort = 8080
```

**Listing C.2** Unencrypted connection detection with OWASP Zap Proxy: Excerpt of build.gradle

```
1     // tag::repository[]
2     repositories {
3           mavenCentral() // "http://repo1.maven.org/maven2"
4
5           // maven {
6           //     url "https://repo1.maven.org/maven2"
7           //}
8
9     }
10    // end::repository[]
```

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

## Project: s

Scan Information :

- *dependency-check version*: 3.1.1
- *Dependencies Scanned*: 291 (202 unique)
- *Vulnerable Dependencies*: 11

| Dependency | CPE | Coordinates | Highest Severity | CVE Count | CPE Confidence | Evidence Count |
|---|---|---|---|---|---|---|
| inspectit-agent.jar: spring-core-3.2.16.RELEASE.jar | cpe:/a:pivotal_software:spring_framework:3.2.16<br>cpe:/a:springsource:spring_framework:3.2.16<br>cpe:/a:pivotal:spring_framework:3.2.16 | org.springframework:spring-core:3.2.16.RELEASE ✓ | Medium | 2 | Highest | 30 |
| [redacted]-1.1.0-SNAPSHOT.jar: tomcat-embed-core-8.5.14.jar | cpe:/a:apache_software_foundation:tomcat:8.5.14<br>cpe:/a:apache:tomcat:8.5.14<br>cpe:/a:apache_tomcat:apache_tomcat:8.5.14 | org.apache.tomcat.embed:tomcat-embed-core:8.5.14 ✓ | Medium | 4 | Highest | 21 |
| [redacted]-1.1.0-SNAPSHOT.jar: jackson-databind-2.8.8.jar | cpe:/a:fasterxml:jackson:2.8.8<br>cpe:/a:fasterxml:jackson-databind:2.8.8 | com.fasterxml.jackson.core:jackson-databind:2.8.8 ✓ | High | 3 | Highest | 38 |
| inspectit-agent.jar: commons-collections-3.2.1.jar | cpe:/a:apache:commons_collections:3.2.1 | commons-collections:commons-collections:3.2.1 ✓ | High | 2 | Highest | 34 |
| [redacted]-1.1.0-SNAPSHOT.jar: spring-security-core-4.2.2.RELEASE.jar | cpe:/a:pivotal_software:spring_security:4.2.2:release | org.springframework.security:spring-security-core:4.2.2.RELEASE ✓ | Medium | 1 | Highest | 26 |
| api-1.1.0-SNAPSHOT.jar: spring-security-crypto-4.2.2.RELEASE.jar | cpe:/a:pivotal_software:spring_security:4.2.2:release | org.springframework.security:spring-security-crypto:4.2.2.RELEASE ✓ | Medium | 1 | Highest | 28 |
| api-1.1.0-SNAPSHOT.jar: netty-codec-4.0.27.Final.jar | cpe:/a:netty_project:netty:4.0.27 | io.netty:netty-codec:4.0.27.Final ✓ | High | 2 | Highest | 26 |
| [redacted]-1.1.0-SNAPSHOT.jar: spring-cloud-cloudfoundry-connector-1.2.3.RELEASE.jar/META-INF/maven/com.fasterxml.jackson.core/jackson-databind/pom.xml | cpe:/a:fasterxml:jackson-databind:2.3.3<br>cpe:/a:fasterxml:jackson:2.3.3 | com.fasterxml.jackson.core:jackson-databind:2.3.3 | High | 3 | Highest | 16 |
| moment:2.15.1 | | moment:2.15.1 | High | 1 | | 3 |
| hoek:2.16.3 | | hoek:2.16.3 | Medium | 1 | | 3 |

**Figure C.1.:** Pipeline A: OWASP dependency check results

# Bibliography


[18] *International Standard ISO/IEC 27000: Information technology–Security techniques–Information security management systems–Overview and vocabulary*. Fith edition. 2018 (cit. on pp. 11, 12).

[AGK15] M. Abomhara, M. Gerdes, G. M. Køien. "A STRIDE-Based Threat Model for Telehealth Systems." In: *Norsk informasjonssikkerhetskonferanse (NISK)*. Vol. 8. 1. 2015, pp. 82–96 (cit. on p. 36).

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. "Basic concepts and taxonomy of dependable and secure computing." In: *IEEE transactions on dependable and secure computing*. Vol. 1.1. 2004, pp. 11–33 (cit. on pp. 13–15).

[And14] J. Andress. *The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice*. 2nd Edition. Syngress, 2014 (cit. on p. 11).

[AS17] A. Ashbel, P. Schmitz. *Datenschutz-Grundverordnung Datensicherheit in der DSGVO*. Ed. by Security Insider. 2017. URL: https://www.security-insider.de/datensicherheit-in-der-dsgvo-a-602299/ (cit. on p. 1).

[Atl] Atlassian, ed. *Bitbucket*. URL: https://bitbucket.org/ (cit. on p. 26).

[AWSa] AWS, ed. *Amazon EC2 Container Service*. URL: https://aws.amazon.com/de/ecs/ (cit. on p. 29).

[AWSb] AWS. *Projekte in AWS - Einrichten eines Jenkins-Build-Servers*. URL: https://aws.amazon.com/de/getting-started/projects/setup-jenkins-build-server/ (cit. on p. 29).

[Bar07] P. Barzin. "Vulnerability of Code." In: *Datenschutz und Datensicherheit-DuD* 31.12 (2007), pp. 884–887 (cit. on pp. 22, 23).

[BBv+01] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. *Manifesto for agile software development*. 2001. URL: http://agilemanifesto.org/principles.html (cit. on p. 1).

[BHR+15] L. Bass, R. Holz, P. Rimba, A. B. Tran, L. Zhu. “Securing a Deployment Pipeline.” In: *IEEE/ACM 3rd International Workshop on Release Engineering*. 2015, pp. 4–7 (cit. on pp. 34, 53, 58).

[Bir16] J. Bird. *DevOpsSec: Securing software through continuous delivery*. O’Reilly Media, 2016 (cit. on pp. 21, 37, 38, 71).

[BM05] S. Barnum, G. McGraw. “Knowledge for Software Security.” In: *IEEE Security and Privacy Magazine*. Vol. 3.2. 2005, pp. 74–78 (cit. on p. 15).

[Car17] K. Carter. “Francois Raynaud on DevSecOps.” In: *IEEE Software*. Vol. 34.5. 2017, pp. 93–96 (cit. on pp. 13, 37).

[CDL16] M. Conti, N. Dragoni, V. Lesyk. “A Survey of Man In The Middle Attacks.” In: *IEEE Communications Surveys & Tutorials*. Vol. 18.3. 2016, pp. 2027–2051 (cit. on pp. 22, 23).

[CDW07] B. Chess, F. DeQuan Lee, J. West. *Attacking the Build through Cross-Build Injection How Your Build Process Can Open the Gates to a Trojan Horse*. Ed. by F. Software. 2007. URL: http://www.diogocatapreta.com.br/wp-content/uploads/2007/10/fortify_attacking_the_build.pdf (cit. on pp. 24, 25).

[Che15] L. Chen. “Continuous delivery: Huge benefits, but challenges too.” In: *IEEE Software*. Vol. 32.2. 2015, pp. 50–54 (cit. on p. 8).

[Clo17] CloudBees, ed. *Jenkins Community Announces Record Growth and Innovation in 2017*. 2017. URL: https://www.cloudbees.com/press/jenkins-community-announces-record-growth-and-innovation-2017 (cit. on p. 71).

[Col14] E. Cole. *Insider Threats in Law Enforcement*. Ed. by SANS Institute, InfoSec Reading Room. 2014. URL: https://www.sans.org/reading-room/whitepapers/analyst/insider-threats-law-enforcement-35402 (cit. on p. 53).

[Col17] E. Cole. *Defending Against the Wrong Enemy:2017 SANS Insider Threat Survey*. Ed. by SANS Institute, InfoSec Reading Room. 2017. URL: https://www.sans.org/reading-room/whitepapers/analyst/defending-wrong-enemy-2017-insider-threat-survey-37890 (cit. on pp. 52, 53).

[Con] Concourse, ed. *Concourse: CI that scales with your project.* URL: https://concourse.ci/ (cit. on pp. 27, 28).

[CVE] CVE, ed. *Common Vulnerabilities and Exposures*. URL: https://cve.mitre.org/index.html (cit. on p. 17).

[Del15] T. Delafuente. *Docker Security Tools: Audit and Vulnerability Assessment*. Ed. by Alfresco Content Services. 2015. URL: https://community.alfresco.com/community/ecm/blog/2015/12/03/ (cit. on pp. 79, 81).

[Deu] Deutsche Telekom, ed. *DevSec Hardening Framework*. URL: http://dev-sec.io/ (cit. on p. 84).

[Dij72] E. W. Dijkstra. "The humble programmer." In: *Communications of the ACM*. Vol. 15.10. 1972, pp. 859–866 (cit. on p. 116).

[DKS16] A. Detlefsen, S. van Koningsveld, M. Smith. *How to Use OWASP Security Logging*. Ed. by OWASP. 2016. URL: https://www.slideshare.net/MiltonSmith6/how-to-use-owasp-security-logging (cit. on p. 86).

[Doc] Docker, ed. *Docker*. URL: https://www.docker.com/ (cit. on p. 28).

[DPL15] A. Dyck, R. Penners, H. Lichter. "Towards Definitions for Release Engineering and DevOps." In: *Proceedings of the Third International Workshop on Release Engineering*. 2015, pp. 3–3 (cit. on p. 10).

[DT16] G. Deepa, P. S. Thilagam. "Securing web applications from injection and logic vulnerabilities: Approaches and challenges." In: *Information and Software Technology*. Vol. 74. 2016, pp. 160–180 (cit. on pp. 33, 34).

[Eck13] C. Eckert. *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. 6., überarbeitete und erweiterte Auflage. Walter de Gruyter, 2013 (cit. on p. 13).

[Fal16] A. Falk. *SecDevOps – Kontinuierlich sichere Software bauen*. Ed. by Inforamtik Aktuell. 2016. URL: https://www.informatik-aktuell.de/betrieb/sicherheit/secdevops-kontinuierlich-sichere-software-bauen.html (cit. on p. 13).

[FI13] J. T. FORCE, T. INITIATIVE. "Security and Privacy Controls for Federal Information Systems and Organizations." In: *NIST Special Publication*. Vol. 800.53. 2013, pp. 8–13 (cit. on p. 13).

[FLF04] K.-J. Farn, S.-K. Lin, A. R.-W. Fung. "A study on information security management system evaluation—assets, threat and vulnerability." In: *Computer Standards & Interfaces*. Vol. 26.6. 2004, pp. 501–513 (cit. on pp. 13, 14).

[FM09] J. Feiman, N. MacDonald. *Magic Quadrant for Static Application Security Testing, Gartner RAS Core Research Note G00205212*. 2009 (cit. on pp. 2, 21, 71).

[Fow06] M. Fowler. *Continuous Integration*. 2006. URL: https://www.martinfowler.com/articles/continuousIntegration.html (cit. on p. 8).

[Fow13] M. Fowler. *Continuous delivery*. 2013. URL: https://martinfowler.com/bliki/ContinuousDelivery.html (cit. on p. 7).

[GHJ13] V. Gruhn, C. Hannebauer, C. John. “Security of Public Continuous Integration Services.” In: *Proceedings of the 9th International Symposium on Open Collaboration*. 2013, 15:1–15:10 (cit. on pp. 16, 36).

[Gil17] H. Gilmore. *DevOps Survey Results: Why Enterprises Are Embracing Continuous Delivery*. Ed. by CloudBees. 2017. URL: https://www.cloudbees.com/blog/devops-survey-results-why-enterprises-are-embracing-continuous-delivery (cit. on p. 1).

[Git] GitLab Inc., ed. *GitLab*. URL: https://about.gitlab.com/ (cit. on p. 27).

[Gue15] Guest Contributor. *THE APPSEC FACEOFF: STATIC ANALYSIS vs DAST vs PEN TESTING*. Ed. by The Cyber Security Place. 2015. URL: https://thecybersecurityplace.com/the-appsec-faceoff-static-analysis-vs-dast-vs-pen-testing/ (cit. on p. 21).

[Has] HashiCorp, ed. *Consul*. URL: https://www.consul.io/ (cit. on p. 29).

[Hay18] S. G. Hays. “KEEPING UP WITH THE CRYPTOS Cryptocurrencies Likely “Best Fraud Show” in 2018.” In: *The receiver - official publication of the National Association of Federal Equity Receivers*. 2018, pp. 1–7 (cit. on p. 25).

[HF11] J. Humble, D. Farley. *Continuous delivery: [reliable software releases through build, test, and deployment automation]*. A Martin Fowler signature book. Upper Saddle River, NJ: Addison-Wesley, 2011 (cit. on pp. 1, 7, 8).

[HKA+14] S. Hussain, A. Kamal, S. Ahmad, G. Rasool, S. Iqbal. “THREAT MODELLING METHODOLOGIES: A SURVEY.” In: *Sci. Int.(Lahore)*. Vol. 26.4. 2014, pp. 1607–1609 (cit. on p. 18).

[HTL+05] Y.-W. Huang, C.-H. Tsai, T.-P. Lin, S.-K. Huang, D. T. Lee, S.-Y. Kuo. “A testing framework for Web application security assessment.” In: *Computer Networks*. Vol. 48.5. 2005, pp. 739–761 (cit. on p. 34).

[HYH+04] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, S.-Y. Kuo. “Securing Web Application Code by Static Analysis and Runtime Protection.” In: *Proceedings of the 13th international conference on World Wide Web*. 2004, pp. 40–52 (cit. on p. 34).

[IBM] IBM, ed. *IBM WebSphere Application Server*. URL: https://www.ibm.com/cloud/websphere-application-platform (cit. on p. 29).

[JCL17] M. G. Jaatun, D. S. Cruzes, J. Luna. “DevOps for Better Software Security in the Cloud Invited Paper.” In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. 2017, 69:1–69:6 (cit. on p. 11).

[Jena] Jenkins, ed. *Jenkins*. URL: https://jenkins.io/ (cit. on pp. 27, 28).

[Jenb] Jenkins, ed. *Using a Jenkinsfile*. URL: https://jenkins.io/doc/book/pipeline/jenkinsfile/ (cit. on p. 27).

[JFr] JFrog, ed. *JFrog Artifactory the world’s only universal artifact repository manager*. URL: https://www.jfrog.com/artifactory/ (cit. on pp. 29, 31, 82).

[JMC09] W. Jimenez, A. Mammar, A. Cavalli. “Software vulnerabilities, prevention and detection methods: A review.” In: *Security in Model-Driven Architecture* (2009), p. 6 (cit. on pp. 2, 15, 16).

[KDWH16] G. Kim, P. Debois, J. Willis, J. Humble. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution, 2016 (cit. on p. 54).

[KKH+07] Y. Kosuga, K. Kono, M. Hanaoka, M. Hishiyama, Y. Takahama. “Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection.” In: *Computer Security Applications Conference (ACSAC), Twenty-Third Annual*. 2007, pp. 107–117 (cit. on p. 34).

[Kuu17] J. Kuusela. “Security testing in continuous integration processes.” Masterthesis. Aalto University, 2017 (cit. on pp. 38, 71).

[Lig09] P. Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. 2. Auflage. Springer Science & Business Media, 2009 (cit. on p. 10).

[Lip17] S. Lipke. “Building a Secure Software Supply Chain.” Masterthesis. Stuttgart Media University, 2017 (cit. on pp. 36, 71).

[LS04] H. Langweg, E. Snekkenes. “A classification of malicious software attacks.” In: *Proceedings of 23rd IEEE International Performance, Computing, and Communications Conference*. 2004, pp. 827–832 (cit. on pp. 16, 21).

[LWC+12] T. Lee, G. Won, S. Cho, N. Park, D. Won. “Detection and Mitigation of Web Application Vulnerabilities Based on Security Testing.” In: *IFIP International Conference on Network and Parallel Computing*. 2012, pp. 138–144 (cit. on p. 33).

[MA10] C. Möckel, A. E. Abdallah. “Threat modeling approaches and tools for securing architectural designs of an e-banking application.” In: *Information Assurance and Security (IAS), 2010 Sixth International Conference on*. 2010, pp. 149–154 (cit. on p. 36).

[Mak12] S. Mak. *Cross-Build Injection Attacks: How safe is your Java build?* 2012. URL: http://branchandbound.net/blog/security/2012/03/crossbuild-injection-how-safe-is-your-build/ (cit. on pp. 24, 25).

[Mar] I. Marković. *OWASP Risk Assessment Calculator*. URL: https://www.security-net.biz/files/owaspriskcalc.html (cit. on pp. 19, 20).

[Mic] Microsoft, ed. *HockeApp*. URL: https://hockeyapp.net/ (cit. on p. 30).

[MIT] MITRE Corporation, ed. *CWE - CWE List Version 2.11: CWE is a Software Assurance strategic initiative sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security*. URL: https://cwe.mitre.org/data/ (cit. on p. 17).

[MLY05] S. Myagmar, A. J. Lee, W. Yurcik. "Threat Modeling as a Basis for Security Requirements." In: *Symposium on requirements engineering for information security (SREIS)*. 2005, pp. 1–8 (cit. on p. 18).

[MO16] V. Mohan, L. B. Othmane. "SecDevOps: Is It a Marketing Buzzword?-Mapping Research on Security in DevOps." In: *Availability, Reliability and Security (ARES), 11th International Conference*. 2016, pp. 542–547 (cit. on p. 13).

[MW05] N. Mack, C. Woodsong. *Qualitative Research Methods: A Data Collector's Field Guide*. North Carolina: FLI and USAID, 2005 (cit. on p. 41).

[Nat] National Institute of Standards and Technology U.S. Department of Commerce, ed. *National Vulnerability Database*. URL: https://nvd.nist.gov/ (cit. on p. 17).

[oA] ozangunalp, Alexandre DuBreuil. *Jenkins Pipeline Unit*. URL: https://github.com/jenkinsci/JenkinsPipelineUnit (cit. on pp. 72, 73).

[OWAa] OWASP, ed. *CISO AppSec Guide: Criteria for Managing Application Security Risks*. URL: https://www.owasp.org/index.php/CISO_AppSec_Guide:_Criteria_for_Managing_Application_Security_Risks (cit. on p. 14).

[OWAb] OWASP, ed. *OWASP Risk Rating Methodology*. URL: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology (cit. on pp. 19, 20).

[OWAc] OWASP, ed. *OWASP ZAP 2.6 Getting Started Guide*. URL: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project (cit. on p. 85).

[OWA17a] OWASP, ed. *Category:OWASP Top Ten Project - OWASP*. 21.10.2017. URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (cit. on p. 16).

[OWA17b] OWASP, ed. *Application Threat Modeling*. 2017. URL: https://www.owasp.org/index.php/Application_Threat_Modeling (cit. on pp. 18, 19, 34, 77).

[OWA17c] OWASP, ed. *OWASP Top 10-2017: The Ten Most Critical Web Application Security Risks*. 2017. URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (cit. on p. 16).

[Pau12] S. Paulus. *Basiswissen Sichere Software: Aus-und Weiterbildung zum ISSECO Certified Professionell for Secure Software Engineering*. dpunkt. verlag, 2012 (cit. on pp. 2, 10–12).

[PD17] Puppet, DevOps Reasearch and Assessment, eds. *State of DevOps Report*. 2017. URL: https://www.ipexpoeurope.com/content/download/10069/143970/file/2017-state-of-devops-report.pdf (cit. on p. 1).

[Pec16] V. Pecanac. *Top 8 Continuous Integration Tools*. Ed. by Code Maze. 2016. URL: https://code-maze.com/top-8-continuous-integration-tools/ (cit. on p. 27).

[Pes08] L. Pesante. "Introduction to Information Security." In: *CERT, Software Engineering Institute*. 2008, pp. 1–3 (cit. on pp. 11, 12).

[Pup] Puppet, ed. *Puppet*. URL: https://puppet.com/ (cit. on p. 29).

[RHRR12] P. Runeson, M. Host, A. Rainer, B. Regnell. *CASE STUDY RESEARCH IN SOFTWARE ENGINEERING: Guidelines and Examples*. John Wiley & Sons, 2012 (cit. on p. 91).

[Run] Rundeck, ed. *Rundeck*. URL: http://rundeck.org/ (cit. on p. 30).

[RZB+15] P. Rimba, L. Zhu, L. Bass, I. Kuz, S. Reeves. "Composing Patterns to Construct Secure Systems." In: *Proceedings of the 2015 11th European Dependable Computing Conference (EDCC)*. 2015, pp. 213–224 (cit. on pp. 7, 34, 35).

[SAZ17] M. Shahin, M. Ali Babar, L. Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices." In: *IEEE Access*. Vol. 5. 2017, pp. 3909–3943 (cit. on p. 7).

[Sch15] C. Schneider. "Security DevOps: Free pentester's time to focus on high-hanging fruits." In: *HackPra Allstars,(Amsterdam, Netherlands)*. 2015 (cit. on pp. 38, 71).

[Sch17] D. Schirrmacher. *Sicherheitsupdates: Jenkins und mehrere Plugins angreifbar*. Ed. by heise Security. 2017. URL: https://www.heise.de/security/meldung/Sicherheitsupdates-Jenkins-und-mehrere-Plugins-angreifbar-3864651.html (cit. on pp. 53, 58).

[SCI] SCIP AG group, ed. *Vulnerability Database*. URL: https://vuldb.com/ (cit. on p. 17).

[SGE17] R. Shu, X. Gu, W. Enck. "A Study of Security Vulnerabilities on Docker Hub." In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. 2017, pp. 269–280 (cit. on pp. 38, 71).

[SGF02] G. Stoneburner, A. Y. Goguen, A. Feringa. "Risk Management Guide For Information Technology Systems." In: *Special publication 800-30.* 2002 (cit. on pp. 19, 20).

[Sho14a] A. Shostack. *Elevation of Privilege: EoP_Card Game Images*. Ed. by Microsoft. 2014. URL: https://www.microsoft.com/en-us/download/details.aspx?id=20303 (cit. on p. 53).

[Sho14b] A. Shostack. *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014 (cit. on pp. 18, 53).

[Siv17] K. Siva Prasad Reddy. *Beginning Spring Boot 2: Applications and Microservices with the Spring Framework*. 2017 (cit. on p. 34).

[Son] Sonatype, ed. *Nexus Repository OSS*. URL: https://www.sonatype.com/ (cit. on p. 30).

[Sta17] D. Stažić. "Security DevOps: Konzeption einer Umgebung zur Integration von Sicherheitstests in aglie Softwareentwicklungsprozesse." Masterthesis. Reutlingen University, 2017 (cit. on p. 39).

[Sto15] A. Storms. "How Security can be the Next Force Multiplier in DevOps." In: *RSAConference,(San Francisco, USA)*. 2015 (cit. on pp. 38, 71).

[Tas02] G. Tassey. "The Economic Impacts of Inadequate Infrastructure for Software Testing." In: *National Institute of Standards and Technology, RTI Project*. 2002 (cit. on p. 18).

[TC08] A. K. Talukder, M. Chaitanya. *Architecting Secure Software Systems*. CRC Press, 2008 (cit. on pp. 11–13, 21–24).

[Tor17] L. Torvalds. *e-mail: Re: [GIT PULL] usercopy whitelisting for v4.15-rc1*. 2017. URL: http://lkml.iu.edu/hypermail/linux/kernel/1711.2/01701.html (cit. on p. 117).

[Tre18] Trend Micro, ed. *Tesla and Jenkins Servers Fall Victim to Cryptominers*. 2018. URL: https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/tesla-and-jenkins-servers-fall-victim-to-cryptominers (cit. on pp. 25, 26).

[URS+17] F. Ullah, A. J. Raft, M. Shahin, M. Zahedi, M. Ali Babar. "Security Support in Continuous Deployment Pipeline." In: *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering*. SCITEPRESS - Science and Technology Publications, 2017, pp. 57–68 (cit. on pp. 34–36, 71).

[UW16] A. A. Ur Rahman, L. Williams. "Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices." In: *Proceedings of the International Workshop on Continuous Software Evolution and Delivery - CSED '16*. New York, USA: ACM Press, 2016, pp. 70–76 (cit. on pp. 13, 37).

[Wol16] E. Wolff. *Continuous delivery: Der pragmatische Einstieg*. dpunkt. verlag, 2016 (cit. on pp. 7, 10).

[Xeb] XebiaLabs Enterprise DevOps, ed. *The Ultimate DevOps Tool Chest*. URL: https://xebialabs.com/the-ultimate-devops-tool-chest/ (cit. on p. 43).

All links were last followed on April 16, 2018.

**Declaration**

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature