Vulnerabilities in Continuous Delivery Pipelines?
A Case Study

Christina Paule
Novatec Consulting GmbH
Leinfelden-Echterdingen, Germany

Thomas F. Düllmann, and André van Hoorn
Institute of Software Technology, University of Stuttgart
Stuttgart, Germany

Abstract—More and more companies are in the process of adopting modern continuous software development practices and approaches like continuous integration (CI), continuous delivery (CD), or DevOps. These approaches can support companies in order to increase the development speed, the frequency of product increments, and the time to market. To be able to get these advantages, especially the tooling and infrastructure need to be reliable and secure. In case CI/CD is compromised or even unavailable, all mentioned advantages are at stake. Potentially, this could also even hinder the forthcoming of the software development. Therefore, our goal was to identify which vulnerabilities are present in industry CD pipelines and how they can be detected. In this paper, we present our results of an industry case study which includes a qualitative survey of agile project teams regarding the awareness of security in CI/CD, the analysis and abstraction of two CD pipelines, and a threat analysis based on the deducted CD pipeline to identify vulnerabilities. In this case study, we found that the team members that work with the CD pipeline in different roles do not have a strong security background but are aware of security attributes in general. Furthermore, two CD pipelines from industry projects were analyzed using the STRIDE threat analysis approach. In total, we identified 22 vulnerabilities that have been confirmed by the project teams.

I. INTRODUCTION

By applying continuous delivery (CD), companies are able to deploy application changes to the customer rapidly and reliably from the software repository to the customer’s hands [1]. Based on the results of a study by Hurwitz & Associates [2], one may assert with confidence that the CD process is an essential component in software development projects.

However, Paulus [3] has shown that, as long as there are no security issues, companies have little interest in application vulnerabilities which could potentially result in a lack of security. Also, they report that insecure applications can severely damage the image of companies. Trend Micro [4] observed that servers are often misconfigured and insecure, and that also CD applications like Jenkins are becoming targets of attacks (e.g., cryptojacking). According to Gruhn et al. [5], CI/CD tools have more vulnerabilities than communicated through vulnerability databases and communities (e.g., OWASP).

This strengthens our claim that industry CD systems need more attention in terms of security. Therefore, the following research question arises:

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

To investigate this question we conducted a case study [6]. With this study, we wanted to get insights into industry CD pipelines, the related project teams, and vulnerabilities in practice. In our case study consisting of two project teams working with CD pipelines in industry, we did the following:

(i) Survey to identify roles team members have regarding the CD pipelines in their projects, their experience with security aspects, and their opinion on the most important security attributes. The intention of the survey was to focus on the most relevant security attributes for the following steps. Even though the survey did not investigate the research question directly, it provided insights into the context CD pipelines are operated in.

(ii) Analysis of two industry CD pipelines focusing on the structure and the overall process. As well as the deduction of an abstracted CD pipeline based on the CD pipelines of the two projects supported by personal interviews with experts for a detailed understanding.

(iii) Execution of a STRIDE threat analysis [7] focusing on the identified most important security attributes (confidentiality, integrity, availability) and mapping of the identified threats based on NIST and OWASP project methodologies for risk severities.

(iv) Manual vulnerability assessment based on the results of the STRIDE threat analysis.

(v) Identification of tools suitable for detection and mitigation of the found threats.

(vi) Validation of results with the project teams.

In total, 22 security vulnerabilities could be identified and confirmed. Furthermore, we could identify tools that potentially could either detect or mitigate such vulnerabilities.

Section II presents foundations on CD pipelines, security, and the STRIDE approach. The different parts of the case study mentioned above are presented, i.e., survey (Section III), CD pipeline analysis and abstraction (Section IV), threat analysis with STRIDE (Section V), vulnerability assessment (Section VI), tool identification (Section VII), and verification of the results (Section VIII). The findings are discussed in Section IX and related work is presented in Section X. The paper is concluded in Section XI. The thesis by Paule [8] is the foundation of this paper and includes detailed resources.

This work is partly funded by the Baden-Württemberg Stiftung (ORCAS Project).

II. Foundations

A. CD pipelines & CD servers

According to Humble et al. [1], a deployment pipeline is “an automated manifestation of your process for getting software from version control into the hands of your users”. In contrast to CD pipelines, CI pipelines do not include the deployment of software, but only the automated preparation of the artifacts. Even though this process does not have to be automated, it is common to use a CD server (e.g., Jenkins) to define the process and execute the steps automatically. So-called stages allow a logical grouping of more fine-grained steps. For example, all the steps that are required for testing the artifact could be grouped in the test stage, which could include the setup of a test environment, the execution of the actual tests, and the tear-down of the test environment. If a stage finished successfully, the next stage is triggered. In case a stage failed, the whole process fails.

B. Security

The main objectives of software security are the attributes of information security (confidentiality, integrity, and availability; also known as the CIA triad) [3], [9] that need to be protected. There are three additional attributes in literature, namely authentication, authorization [10], and non-repudiation [11].

The NIST defines a vulnerability as a “weakness in an information system [...] that could be exploited or triggered by a threat source” [12]. Vulnerabilities (e.g., programming mistakes or software errors) can occur anywhere in software [10]. According to Farn et al. [13], vulnerabilities are connected with threats, assets, values, and risks.

C. STRIDE threat modeling and analysis

Threat modeling is an approach to detect potential vulnerabilities, further risks, and threats in an application as early as possible [14]. Threat modeling extracts the components of a system and considers the possible entry points from an attacker’s point of view [14].

Out of various threat modeling approaches, the STRIDE threat modeling method is the most widely used method [15]. As reported by OWASP, threat modeling is a process that can be executed in three steps [14]. In the first step, the application has to be understood with all its components and connections. The second step is to identify the threats and vulnerabilities in the application. Threats can be assigned to six different categories according to STRIDE, namely spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege [7]. In the third step, the results are assessed. In this research, this assessment is done with the OWASP risk rating methodology [14], the calculator of Maković [16], and the definition of the NIST [17]. As a result, the overall risk severity is the product of the likelihood and an impact level. For example, a higher likelihood factor is given if the attacker needs no resource access rights to exploit the vulnerability. If the attacker needs full access rights and detailed system knowledge then a low likelihood factor exists because it is unlikely that an attacker can exploit the vulnerability. The damage to the company and the amount of data which is gained through the exploitation of the vulnerability defines the impact factor. Table I shows that the overall risk severity can have a level between none (no risk) and critical [14].

III. Survey

Mack et al. [18] recommends a qualitative research method to understand the present problem or the context of a topic. In order to better understand the context and narrow down the security attributes to look at, we conducted a survey. It was conducted in project teams working with CD pipelines in a DevOps context. The survey was set up as an online survey that was sent via e-mail to approximately 100 employees of a selected company which includes the investigated CD pipeline project teams of the case study. The participation was voluntary and was possible at any time. In total, there were 19 completed surveys containing nine questions about the roles of the participants regarding CD pipelines, the participants’ opinion on the security of CD pipelines and the most prevalent problems, and their background on security and tooling for CD pipelines. The survey details and results can be found in the thesis by Paule [8].

Question 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.

We manually aggregated the objectives into the following categories, presented in no specific order: “Securing source code, logs and artifacts”, “Build steps should not be manipulated”, “No pipeline modification through unauthorized persons”, “No triggering of the pipeline through unauthorized persons”, “Securing environment properties such as login data”, “Reduce human errors (storing password)”, “Securing credentials (encrypt all sensitive data)”, “No vulnerabilities in dependencies”, “Secure transmission over HTTPS or SSH”, “Use 4-eye-principle”, “Check access rights of the components of the CD pipeline”.

Question 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, non-repudiation) according to their importance. The attribute on top is the most important for you.

Out of the six ranked attributes, the top three are used for scoping the threat analysis. The results of this question help to delimit the subject because it reflects the interest and the

<table>
<thead>
<tr>
<th>Likelihood</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impact</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>

TABLE I: Overall risk severity matrix based on OWASP [14]
necessity from the viewpoint of the employees. In the view of the employees, the three top-ranked attributes are integrity, availability, and confidentiality.

**Question 3:** In your opinion what are possible attack scenarios for the pipeline you use? Against which attacks would you like to protect your pipeline?

Based on the results from question 2, we narrowed down the classification to the three security attributes of confidentiality (C), integrity (I), and availability (A). All answers that could not be assigned to any of these, were labeled with "N/A".

The answers and the results of the classification can be found in Table II on the following page. Only typographical errors were corrected in the listed answers.

Seven answers could be assigned to attack scenarios related to confidentiality, 15 answers to integrity, and five to availability. There were 27 answers in total that could, in parts, be assigned to multiple security attributes.

**Question 4:** Which security objectives are pursued in your project in respect to CD pipelines? Which are implemented?

The security objectives the participants mentioned are: requiring authentication and authorization, securing credentials and hiding critical data, reviewing the process, no information should be included in the source code of applications, access control, keeping the pipeline components and software up to date.

It can be seen that in the industrial projects, authentication and authorization are implemented. In addition, securing sensitive data and access rights also contribute to secure the pipeline. If a project does not pursue security goals, it cannot be guaranteed to the customer that the software will be deployed securely.

**Question 5:** How many years of experience in software development do you approximately have?

The results show that the participants have 10.32 years of development experience on average.

**Question 6:** Which tools do you know and/or use? Response options: (DevOps tools) Jenkins; Kubernetes; TeamCity; Spinaker; 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

The answers, which can be found in the thesis by Paule [8], show that the CI server Jenkins is known by all 15 participants. The know-how in security testing tools is not as well-known as static analysis tools. In summary, it can be said that the competence of the company lies in the CI tool Jenkins and in static analysis tools.

**Question 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.

The results show that 58% of the 19 participants interact with all facets of a pipeline (using, installing, configuring, and operating the pipeline). 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 95% of the participants come into frequent contact with the pipeline.

**Question 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)

With a median of four, the results show that the employees realize that it is necessary to do research on this topic.

**Question 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 answers show that most employees only occasionally deal with security topics during their development process. Five employees deal with the security context quite often and one is never concerned with it.

**Question 10** (Only for the participants in the teams of the analyzed CD pipelines): In the next step think about the security of the <project name> CD pipeline. In your opinion how secure is this pipeline? Response options: 1; 2; 3; 4; 5 (1: Many threatening vulnerabilities; 5: No vulnerabilities)

For both teams, the median value of the answers was 3. This shows, that there is neither the expectation that no vulnerabilities exist at all nor that there are many.

IV. CD PIPELINE ANALYSIS AND ABSTRACTION

In the following, we follow established steps for designing and planning a software engineering case study [6].

The rationale for and the objective of the study are that the case study investigates the security level of the CD pipelines of a selected software consulting company referred to as Alpha and Beta in this paper, while the teams working with these CD pipelines are organizationally separated and the CD pipelines are related to different customers. The CD pipeline details were identified by oral interviews with the developers.

The goal of the case study is to gain knowledge about how team members assess the security level of the CD pipelines, which security objectives are implemented in the CD pipelines, and which overall risk severity the industrial CD pipelines have. As methods of data collection, a qualitative survey and interviews are used. The additional question of the survey is used to find out how the team members assess the security level. They have the response options 1; 2; 3; 4; 5 (1: means CD pipeline is insecure; 5: means CD pipeline is secure (pipeline has no vulnerabilities)). The vulnerabilities of each CD pipeline were assessed either manually or with tools.

The methods of data analysis comprise collecting vulnerabilities and assess their likelihood and impact levels which lead to an overall risk severity.
A. Industry CD pipelines

We inspected two CD pipelines (Alpha and Beta) that are actively used in industry projects.

1) CD pipeline Alpha: Figure 1 depicts the rough structure of CD pipeline Alpha. At the time of the investigation, the components of the CD pipeline were Bitbucket 4.14.2, Jenkins 2.89.3, Amazon Web Service (AWS), JFrog Artifactory 5.8.3, Rundeck 2.10.2-1, HockeyApp store, and Puppet. The Jenkins master can only be accessed from the company’s intranet. In an external Cloud Foundry Bitbucket, JFrog Artifactory, and a Rundeck server are installed. Puppet is used to set up the deployment environment.

The first step of pipeline Alpha is that a developer commits code changes to Bitbucket (1). After a developer has reviewed that the code has no issues, it can be committed. Jenkins triggers a new instance of the CD pipeline (2, 3). The 20 available EC2 container instances in the AWS cloud are used to build the project. The build phase is successfully completed when all tests (e.g., JUnit) passed. The built artifact is stored in the JFrog Artifactory (6). For every testing type (automatic acceptance test, load testing), a separate pipeline is triggered. The deployment phase is triggered manually. Jenkins notifies the Rundeck server which deployment environment (e.g., production) it should use. The aim of the deployment step is to deploy the desired artifact to Cloud Foundry or, if it is an Android app, to the HockeyApp store (7, 8, 9).

The team of CD pipeline Alpha has specified security objectives that comprise performing regular updates, restricting access of users, usage of authentication mechanisms, no or restricted access from outside to the CD pipeline infrastructure.

2) CD pipeline Beta: Figure 2 depicts the rough structure of the investigated CD pipeline Beta. At the time of the investigation, the components of the CD pipeline are GitLab Community 10.1.0, Jenkins 2.32.2, Sonatype Nexus Repository OSS 2.14.3, IBM WebSphere 8.0.0.10, SFTP server, and HashiCorp Consul 0.75.5. All components of the CD pipeline can only be reached over the customer intranet. The results of each CD pipeline stage are stored in a key-value database which is provided by HashiCorp Consul. The first step of the CD pipeline is that the developer commits her code changes to Bitbucket (1). After a developer has reviewed that the code has no issues, it can be committed. Jenkins triggers a new instance of the CD pipeline (2, 3). The 20 available EC2 container instances in the AWS cloud are used to build the project. The build phase is successfully completed when all tests (e.g., JUnit) passed. The built artifact is stored in the JFrog Artifactory (6). For every testing type (automatic acceptance test, load testing), a separate pipeline is triggered. The deployment phase is triggered manually. Jenkins notifies the Rundeck server which deployment environment (e.g., production) it should use. The aim of the deployment step is to deploy the desired artifact to Cloud Foundry or, if it is an Android app, to the HockeyApp store (7, 8, 9).

The team of CD pipeline Beta has specified security objectives that comprise performing regular updates, restricting access of users, usage of authentication mechanisms, no or restricted access from outside to the CD pipeline infrastructure.
changes to GitLab (1). Jenkins triggers a new instance of this CD pipeline (2). After that, the application is built on two Jenkins nodes (3). The build stage ends successfully if all tests have been passed (e.g., SonarQube). The artifacts are uploaded in the Sonatype Nexus Repository OSS (4). For testing, the application has to be installed on an IBM WebSphere application server (5). The production stage is triggered manually. If this is done, the artifacts are deployed on an SFTP server (6).

The team of CD pipeline Beta has not explicitly specified security objectives but nonetheless implemented security measures like regular updates for components and plugins of the CD pipeline, limited access to CD pipeline components (i.e., intranet only), and restricted access and authentication with limited permissions.

B. Abstracted CD pipeline

A generalized version of the investigated CD pipelines Alpha and Beta is illustrated in Figure 3. The first pipeline’s process step is that a developer pushes source code changes to a source code repository (1). This event notifies the CI/CD server (e.g., using a webhook) (2). The CI/CD server then triggers the first stage ("build") in the CD pipeline process (3). The "build" stage comprises checking out the source code from the source code repository (4), retrieving third-party libraries from the library store (e.g., Maven Central) (5) for building the application and storing the resulting artifacts in an artifact repository (6). If the build stage was successful, the test stage is triggered (7), which executes tests on the artifacts (e.g., JUnit). Afterwards, the deploy stage is triggered (8), which retrieves the latest successfully built artifact from the artifact repository (9), and deploys it to a deployment server (10).

V. THREAT ANALYSIS

To detect vulnerabilities in the abstracted CD pipeline, the aforementioned threat modeling approach STRIDE [19] is applied. Because of the survey results and the three main attributes (CIA triad) from literature, we focused the threat analysis on the threats in the categories tampering (T), information disclosure (I) and denial of service (D) of the STRIDE method. These are investigated for each component and data flow of this CD pipeline. Tampering hurts the attribute integrity and means that the attacker modifies (sensible) data for which she has no authorization. Confidentiality is hurt by information disclosure which means that the software/application displays information to unauthorized persons/attackers. The third attribute availability is hurt by a denial of service attack which means that the attacker slows down the application, causes it to crash, or fills the memory.

In order to consider all possible threats, the cards of the Elevation of Privilege Threat Modeling Card Game [19], which are issued by Microsoft, are used as a basis. For each identified component from the generalized CD pipeline, the aspects of tampering, information disclosure, and denial of service were considered to find possible vulnerabilities. This resulted in 21 tables each referring to a component and one aspect of T, I, and D with each table containing one or multiple possible threats, their effect, and the corresponding vulnerability. The STRIDE tables can be found in the thesis by Paule [8].

These results were mapped to the CD pipelines Alpha and Beta which resulted in 11 vulnerabilities for each of the two CD pipelines with different risks (see Table III and Table IV).

---

1https://search.maven.org/
VI. VULNERABILITY ASSESSMENT

Eventually, we identified the following set of general vulnerabilities that were the result of manual analysis based on the results of the STRIDE analysis:

1. Employees that may harm the security by introducing problems either intentionally or unintentionally,
2. Uncrypted connections between components of the CD pipeline,
3. Insecure environments of the CD pipeline components in terms of their deployment,
4. None or few access restrictions,
5. The use of vulnerable versions of the CD pipeline components,
6. Vulnerable CD pipeline configurations,
7. Vulnerable code commits, CD pipeline scripts, Docker images/containers, artifacts, and
8. No review of changes to the CD pipeline.

VII. TOOL IDENTIFICATION

Based on the identified vulnerabilities, we looked for tools that could help to either detect or mitigate them. We found tools that supposedly are able to satisfy these requirements either for single or even multiple of the CD pipeline stages.

Of course, the tools to apply highly depend on the technologies and steps used in a CD pipeline. The list of identified tools can be found in the thesis by Paule [8].

VIII. RESULT VERIFICATION

After having identified potential vulnerabilities, the project teams were notified of the results. The teams operating the CD pipelines were able to confirm all found vulnerabilities and plan to resolve as many of them as possible. According to them, there might be vulnerabilities that need further discussion with the customer to make all needed changes. Some critical vulnerabilities were fixed immediately.

IX. DISCUSSION

The results show that only the project team working with CD pipeline Alpha has declared explicit security objectives for their CD pipeline. The project team working with CD pipeline Beta indirectly pursues security objectives but their main aim is automation and fast velocity of the delivery process. All in all, both teams try to fulfill their objectives but the project team working with CD pipeline Alpha does this with more awareness. Both teams cannot fulfill the security objectives completely, also 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 are limited in budget and time for it or components are managed by a third party.

The investigation of the CD pipelines show that vulnerabilities exist in both CD pipelines which pose potentially high threats.

In the end, the results were presented to the project teams which confirmed the found vulnerabilities. According to the teams, several of these vulnerabilities could be fixed by now. This shows, that the awareness of vulnerabilities can solve at least some of the potential threats. Though, some may be harder to resolve as third parties need to be involved (e.g., customer) or can hardly be changed within a reasonable time.

Threats to validity

This section discusses threats to validity for our case study.

Internal validity. The case study was conducted in a single company, which might pose a threat to validity. However, the participants of the survey came from different organizational units (i.e., teams). The CD pipelines that were investigated are handled by different teams from the same company and belonged to different customers. It needs to be emphasized that it is difficult to get insights into real-world practices and CD pipelines in the industry, whereas our exploratory work can provide a first glimpse.

Construct validity. The details of the CD pipelines were mainly deduced from information obtained in interviews. A possible threat to validity could be that this information was not completely accurate or not transferred correctly. Even though the STRIDE approach helps to structure the steps of threat analysis, aspects might have been overlooked.

External validity. As we only had a small sample size in the survey (19) as well as for the CD pipelines (two), the results are not generalizable to all industry CD pipelines. However, this exploratory work gives first insights into possible problems in practice that could be further investigated.

X. RELATED WORK

A CD pipeline consists of tools that are in most cases web-based applications or services (e.g., Jenkins). Approaches were developed to detect vulnerabilities in such applications [20–24]. The OWASP Top 10 list 2017 [14] postulates which vulnerabilities occur in most web applications. In comparison to our work, these tools and approaches do not specifically focus on the CD pipeline context.

DevSecOps (Secure DevOps) [25], [26] tools help to automate security in the CD process. Rahman and Williams [25], Bird [27], Schneider [28], Storms [29], and Kuusela [30] presented tools and methods which detect vulnerabilities in applications, partly also inside a CD pipeline. In addition to the mentioned papers, Stažić [31] addressed the addition of security to agile development processes. Despite this work addresses vulnerabilities in the context of CD pipelines, they focus on artifacts that are built by the pipeline, while our work focuses on the pipeline itself in an industry context.

Several approaches have been developed in the form of tactics to increase the security level of CD pipelines [32–34]. In a proof of concept, they mention that they want to eliminate one class of attack vectors with visualization in public CI services. Abomhara et al. [35] investigated threats in a telehealth system and Lipke [36] used the STRIDE method to investigate threats for a CD pipeline based on Docker. In contrast to our work, this work did not apply their findings to real-world industry cases.

XI. CONCLUSION

By conducting a survey and inspecting two CD pipelines from industry using STRIDE, we found that the security of CD pipelines does not have high priority in development teams. Additionally, though most of the team members have
access to the CD pipeline configuration, the lack of security awareness and background in the teams pose a risk to this increasingly business-critical development tool, both in terms of infrastructure, as well as in terms of application.

The goal was to find out how secure specific CD pipelines in a company are. In order to answer this, a case study was conducted. 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. Some vulnerabilities in these CD pipelines occur because the project teams are dependent on the customers’ infrastructure. For both investigated CD pipelines the project teams have to trust in the security of the foreign infrastructure and their own team members. The research shows which vulnerabilities are present in CD pipelines and how they can be detected.

Our findings could improve the awareness of security issues in the teams, but as the circumstances (e.g., project focus, budget, influence on third-party infrastructure) in a business context may be complicated, it might not always be easy to overcome all of the said issues. In order to address security issues in CD pipelines, we aim to investigate ways (e.g., failure injection) to identify dependability and security issues as described in Dullmann et al. [37].

Future work

Since only two industry CD pipelines were investigated, further CD pipelines could be analyzed to validate our findings and to identify additional vulnerabilities in existing CD pipelines. A more extensive case study with a bigger sample size from different companies and multiple projects could show whether the CD pipelines also show similar vulnerabilities and similar patterns in team member background towards security. Another opportunity may be to derive a CD pipeline metamodel (e.g., based on the presented abstracted CD pipeline) to better describe security properties on an abstract level.

REFERENCES