Bachelor’s thesis

Information and Communications Technology

2025

Thien Dinh

Ensuring Security in Continuous Intergration and Deployment Pipelines

Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2025 | 43

Thien Dinh

Ensuring Security in Continuous Intergration and Deployment Pipelines

This thesis aims to integrate security practices into the Stand-Alone and Continuous Integration Deployment (CI/CD) pipelines automation), to prevent issues such as insecure dependencies, secret leakage, or misconfiguration. The research analyzes literature, frameworks, and builds a secure CI/CD pipeline using tools from AWS and open-source scanners such as SonarQube and OWASP Dependency Check.

The implementation was evaluated on a back-end web application featuring React, Node.js, and MySQL. Security was integrated into the build and deployment phases, as well as defined in this study's proposed evaluation metrics. The analysis identified over 20 outdated package vulnerabilities, critical issues with hard-coded secrets. The security tools did introduce some performance overhead (~15% drop in efficiency), but not enough to hinder the deployment.

These results suggest that SMEs and smaller teams can effectively implement a DevSecOps approach if proper planning and tools are utilized, as demonstrated in this research. Such a model for the security pipeline enables reusable and scalable structures to be positioned for the protected software delivery in cloud environments.

Keywords:

CI/CD Security, DevSecOps, Continuous Integration, Continuous Deployment, Supply Chain Attacks, Secrets Management, Vulnerability Assessment, Secure Software Development, AWS Security

Opinnäytetyö (AMK / YAMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2025 | 43

Thien Dinh

Turvallisuuden varmistaminen jatkuvan integraation ja käyttöönoton putkissa

Tämän opinnäytetyön tavoitteena on integroida tietoturvakäytäntöjä Stand-Alone- ja Continuous Integration Deployment (CI/CD) -putkiin (automaatio), jotta voidaan estää ongelmia, kuten turvattomat riippuvuudet, salaisuusvuodot tai virheelliset konfiguraatiot. Tutkimuksessa analysoidaan kirjallisuutta ja kehyksiä ja rakennetaan turvallinen CI/CD-putki käyttämällä AWS-työkaluja ja avoimen lähdekoodin skannereita, kuten SonarQube ja OWASP Dependency Check.

Toteutusta arvioitiin taustapohjaisella verkkosovelluksella, jossa käytettiin Reactia, Node.js:ää ja MySQL:ää. Tietoturva integroitiin rakennus- ja käyttöönottovaiheisiin sekä määriteltiin tässä tutkimuksessa ehdotetuissa arviointimittareissa. Analyysissä tunnistettiin yli 20 vanhentunutta pakettihaavoittuvuutta ja kriittisiä ongelmia kovakoodattujen salaisuuksien kanssa. Tietoturvatyökalut aiheuttivat jonkin verran suorituskykyyn liittyvää ylimääräistä kuormitusta (~15 %:n tehokkuuden lasku), mutta eivät riittävästi haitatakseen käyttöönottoa.

Nämä tulokset viittaavat siihen, että pk-yritykset ja pienemmät tiimit voivat tehokkaasti toteuttaa DevSecOps-lähestymistavan, jos käytetään asianmukaista suunnittelua ja työkaluja, kuten tässä tutkimuksessa on osoitettu. Tällainen tietoturvaputken malli mahdollistaa uudelleenkäytettävien ja skaalautuvien rakenteiden sijoittamisen suojatun ohjelmiston toimitukseen pilviympäristöissä.

Asiasanat:

CI/CD-tietoturva, DevSecOps, Jatkuva integraatio, Jatkuva käyttöönotto, Toimitusketjuhyökkäykset, Salaisuuksien hallinta, Haavoittuvuuksien arviointi, Turvallinen ohjelmistokehitys, AWS-tietoturva.

Content

List of abbreviations (or) symbols 8

1 Introduction 9

1.1 Background and motivation 9

1.2 Problem statement 9

1.3 Research objectives 9

1.4 Scope and Limitations 10

2 Security Challenges and Best Practices in CI/CD Pipelines 11

2.1 Security challenges in CI/CD 11

2.2 Existing security measures and best practices 12

2.3 Case studies on secure CI/CD implementations 13

3 Threat Modeling and Standards 14

3.1 Threat Modeling for CI/CD Pipeline 14

3.2 Relevant Standards and Guidelines 15

4 Role of DevSecOps and Implementation Challenges 17

4.1 Role of DevSecOps in CI/CD 17

4.2 Challenges in Implementation Secure CI/CD 17

5 Security Testing Techniques in CI/CD Pipelines 19

5.1 Static Application security Testing (SAST) 19

5.2 Dynamic Application Security Testing (DAST) 20

5.3 Sofware Composition Analysis (SCA) 21

5.4 Infrastructure as Code (IaC) Scanning 22

6 Secure Supply Chain Management for CI/CD 24

6.1 Dependency Scanning 24

6.2 Artifact Signing and Verification 25

6.3 Provenance Tracking 26

6.4 Least Privilege and Access Control 27

7 Research methodology **28**
7.1 Research approach 28
7.2 Data collection methods 28
7.2.1 Case Study Analysis 29
7.2.2 Practical Implementation and Testing 29
7.3 Evaluation Metrics for CI/CD Security 30
8 Imlementation and Practical Testing **31**
8.1 Goal and Scope 31
8.2 Environment Setup 31
8.3 CI/CD Pipeline Construction 32
8.4 Data Collection and Metrics 33
8.5 Backend Deployment to AWS 34
8.5.1 Launch EC2 Instance 34
8.5.2 Connect to EC2 and Install Dependencies 35
8.5.3 Clone the Project and Run Backend 35
8.5.4 Test API 35
8.6 Observations 35
9 Conclusion **39**
References **40**

Figures

Figure 1. CI/CD pipeline architecture. 32
Figure 2. Connect to EC2. 35
Figure 3. Run the Project. 35
Figure 4. Successful CI/CD Deployment Message. 36
Figure 5. Summary of Security Scanning Results. 37

List of abbreviations (or) symbols

API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration / Continuous Deployment
CVE	Common Vulnerabilities and Exposures
DAST	Dynamic Application Security Testing
DevSecOps	Development, Security, and Operations
EC2	Elastic Compute Cloud
IaC	Infrastructure as Code
IAM	Identity and Access Management
OWASP	Open Worldwide Application Security Project
S3	Simple Storage Service
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SME	Small and Medium-sized Enterprise

1 Introduction

1.1 Background and motivation

In the contemporary world of automation, every step in the software development life cycle is becoming automated, starting from building to deploying the product for end user usage. Integrating automation with software development and implementation has lead to the invention of new terms like DevOps and CI/CD pipelines (Fitzgerald & Stol, 2017). Automation brings great benefits in efficiency while trimming the time required for manual tasks. With the increase in automation, the vulnerability that software systems face from external threats also increases.

1.2 Problem statement

The deployment of further advanced tools that allow organizations to keep up with the increased complexity tends to magnify the existing vulnerabilities even further. New forms of threats to software effortlessly bypass the existing security. Additionally, secrets being exploited within the source code and configuration errors pose major security threats to the entire system (OWASP, n.d.-a). Even with ample security measures, companies come across roadblocks like limited understanding of the resources, insufficient funding, and the difficulty that comes with integrating complex infrastructure.

1.3 Research objectives

This thesis seeks to analyze the existing literature on the topic of CI/CD pipelines and their security implementations, then use that information to outline the most significant gaps in security vulnerabilities and design feasible solutions for the gaps.

Moreover, this thesis works towards focus goal on secure DevOps practices. The objective in undertaking this particular project is to develop deeper competencies and knowledge concerning security in software development life cycle processes or pipelines.

1.4 Scope and Limitations

Focusing on a practical evaluation using open source and AWS-based tools such as AWS CodePipeline, AWS CodeBuild, SonarQube, and OWASP Dependency-Check, this study aims to identify and address security concerns unique to CI/CD setups.Among the limitations are: security in manual or non-automated deployment processes is not within the scope of the study, the findings may not be fully applicable to CI/CD deployments outside of AWS environments.

2 Security Challenges and Best Practices in CI/CD Pipelines

This chapter incorporates a security-centric survey of research focused on the Continuous Integration and Continuous Deployment (CI/CD) systems). It begins with an overview of the prominent security issues associated with the practices of CI/CD. After that, the chapter analyzes best practices in the industry and documents case studies from major technology companies that utilize secure methodologies in their CI/CD practices.

2.1 Security challenges in CI/CD

The deployment of any modern software is now accompanied by sophisticated tools which increase the risk of cyberattacks. When combining multiple tools and services into a single CI/CD pipeline, this increases the risk to the organization's security posture, including vulnerability dependencies and misconfigurations (OWASP, n.d.-b). As the development and operations aspects of a company improve, new vulnerabilities come to light which always need heightened awareness, and practices. Below are some issues to look out for:

Supply Chain Threats: Attackers enslaving third-party dependencies into creating additional malicious code into builds (OWASP, n.d.-g).

Secret Key Management: Continuously expired versions of secrets in codes which are up for public access pose huge setbacks to security (OWASP, n.d.-g).

Misconfigurations: Basic or badly looked after parameters on tools, especially, pipelines, might aid in granting access to hackers (OWASP, n.d.-g).

Inadequate Monitoring: It is near impossible to detect infringements with no efficient logs in place (OWASP, n.d.-g).

Lack of Integrated Security Testing: Security is usually overlooked in favor of meeting deadlines which leads to fragile code being deployed (OWASP, n.d.-g).

2.2 Existing security measures and best practices

To mitigate prevalent security vulnerabilities in CI/CD pipelines, organizations have integrated various tools into their workflows. The following checklist summarizes main activities that strengthen pipeline security: A variety of security measures have been added to their CI/CD workflows to address specific issues.

The term “shift-left security” relates to performing security testing at the early stages of the software development lifecycle, particularly during the design and building phases, as opposed to after deployment. To allow better vulnerability discovery and remediation during the building stage, SAST and DAST tools may be integrated into the initial processes of the CI/CD pipeline. This approach is taken to optimize the development process and reduce the probability of security issues (OWASP, n.d.-a).

Automated Security Scouring: Indiscriminate vulnerability assessment is when the source code is scanned and software code is scanned through the use of tools like OWASP Dependency-Check, SonarQube, and Snyk without any restrictions (OWASP, n.d.-e; SonarQube Docs, n.d.).

Secrets Management Service: Protecting confidential information includes AWS Secrets Manager, HashiCorp Vault, or GitHub Actions Secrets (AWS, n.d.).

Role Based Access Control: Restricting user access to Restricted Based Access With Least Privilege in the CI/CD environment.

Incident and anomalous activity responses alongside Systematized logged activity: Monitoring and recording sundry activities with CloudTrail, and Prometheus, or using the ELK Stack for AWS to log arbitrary activities is

regarded as the ‘uncovering’ of dubious or unauthorized actions (Ehrman, 2023).

2.3 Case studies on secure CI/CD implementations

Numerous companies have been able to successfully add security features to their CI/CD pipelines. Case studies illustrate the real life application of secure CI/CD processes:

Netflix: With a DevSecOps approach, security checks like automated vulnerability scans are performed throughout the CI/CD pipeline (Netflix TechBlog, n.d.).

Google: Integrates principles of BeyondCorp for securing access control and authentication in CI/CD environments (Google Cloud, n.d.).

Microsoft: Follows Azure DevOps Security Best Practices by applying security at every step of the pipeline via policy and compliance automation (Microsoft, 2025).

These case studies confirm how effective a security-centric approach in the strategy of a security-first CI/CD software development processes is to risks management and reliability of software (NS, 2025).

3 Threat Modeling and Standards

To safeguard a CI/CD pipeline properly, context and situation-specific threats need to be understood and predicted first (OWASP, n.d.-c). The chapter talks about relevant security frameworks and standards and some of the most important threat modeling techniques developed for CI/CD environments. These pillars are meant to ensure that compliant and effective policies are implemented around configurations of the pipelines.

3.1 Threat Modeling for CI/CD Pipeline

As a step-by-step process of threat identification and escalating mitigation strategies, threat modeling is a vital approach when it comes to security concerns. With regards to CI/CD pipelines specifically, threat modeling allows the identification of possible gaps in the software delivery process. Using STRIDE (Microsoft, 2022) and MITRE ATT&CK (MITRE, 2023) frameworks allows for proper systematic categorization of the threats.

Some of the identifiable risks include unwanted access to repository databases, code injection during the build phase, and artifact modification post-build. These scenarios represent risks that can be avoided if the proper controls are put place beforehand. Not building a threat model during the design phase of a CI/CD system severely limits the security controls to be unfavorably designed (OWASP, n.d.-h).

An illustration of a STRIDE model usage in a CI/CD context may include a review of each step of the pipeline for certain categories of threats. For mitigating spoofing threats in the “Source” stage, it is possible to misuse Git access to result in tampering, if build scripts are not secured. In the “Build” and “Deploy” stages, threats due to scope creep can include elevation of privileges, exposing secrets, or inadvertent information disclosure via IAM roles.(Microsoft, 2022)

It is encouraged that teams conduct regular threat modeling sessions especially when new tools are added or the pipeline is rearranged (Microsoft, n.d.). Such sessions can be hands-on and should aim for participation from the developer side, DevOps, and security analyst side so that risk assessment coverage is holistic. Threat models must be documented and revisited because they allow the CI/CD pipeline to cope with attempts to manipulate it while aligning with optimal sophisticated security measures (OWASP, n.d.-h).

3.2 Relevant Standards and Guidelines

Several documents have been published citing steps to approach the concern of security in automated software pipeline (ISO, 2013). The most important of these is:

NIST SP 800-204A: Discusses the security principles for microservices architectures with focus on embedding security in CI/CD (NIST, n.d.).

OWASP CI/CD Security Guidelines: Measures to secure code repositories, build & deploy servers, and the overall environment of a pipeline are all covered (OWASP, n.d.-b).

CNCF Security Whitepaper: Deals with approaches for securing the software supply chain in cloud native environments (Moore Marina, 2024).

These standards provide suggestions but also checklists and architectural practices that can be applied(Chandramouli, 2019). As an example, NIST SP 800-204A cites the sealing of identity management, automated vulnerability scanning, and audit logging as necessary within the borders of each step in the CI/CD pipeline. This is advantageous to those working with microservices and cloud-native automation frameworks since those systems are so fluid and distributed (Chandramouli, 2019).

On the other hand, the OWASP CI/CD Security Guidelines offer detailed guidelines relevant for each stage of the pipeline (OWASP, n.d.-a). For example, within the “Build” phase, OWASP recommends that access to build

environments be networked in such a way as to mitigate lateral movement. Moreover, secret management as well as secure storage and least privilege at all levels of CI/CD are also strongly advocated (OWASP, n.d.-a).

The CNCF Security Whitepaper is another example. It extends cloud native security guidelines to container deployment practicalities(Moore Marina, 2024). It focuses on securing artifacts and enforcing the governance of provenance tracking, as well as the implementation of policy-as-code in CI/CD. This is critical to automation in control-heavy, high-traceability systems such as Kubernetes.

4 Role of DevSecOps and Implementation Challenges

DevSecOps capture the most recent shift of incorporating security within DevOps. This chapter discusses the fundamentals of DevSecOps and how it is used to automate security functions in CI/CD pipelines. It also covers why most organizations face difficulties in incorporating security in automated workflows, especially in resource constrained environments.

4.1 Role of DevSecOps in CI/CD

DevSecOps underscores the importance of security right from the start of the software development lifecycle, rather than placing it within the pipeline as an afterthought. In the context of CI/CD, it is the responsibility of DevSecOps to automate the enforcement of security validations, vulnerability scanning, and compliance checks at every stage of the pipeline (Red Hat, 2023).

Some fundamental practices of DevSecOps are (Red Hat, 2023):

Security scanning of code and dependencies is done automatically.

Sensitive data protection for secrets management.

Policy enforcement for Infrastructure as Code (IAC)

Anomaly detection via real time activity monitoring of the pipeline.

4.2 Challenges in Implementation Secure CI/CD

Regardless of the benefits, securing CI/CD pipelines poses a significant (Fitzgerald & Stol, 2017):

Tool Integration Complexity: Automation within a pipeline does not bring easy integration with security tools.

Performance Trade-offs: Security checks during build and deployment significantly elongate the process.

Cultural resistance: Many developers consider security policies to be unnecessary shackles.

Resource Constraints: Less sophisticated firms could lack the expertise or financial resources to implement complex security measures.

The problems with integrating security tools within the CI/CD pipelines mostly arise from the processes themselves and the absence of a unified integration framework. For example, Jenkins and GitHub Actions host hundreds of plugins, but implementing Dependency-Check OWASP, Snyk, or even SonarQube may need bespoke coding and upkeep (Red Hat, 2023). Exposing security measures as integrations means they are not organically embedded into the system, which is frequently the case with fragmented pipelines.

Lack of technological synergy raises severe problems with efficiency bandwidth. The presence of security scanners in the pipeline may hinder deployment cycles, particularly in large projects containing hundreds or thousands of dependencies or microservices (Dewhurst, n.d.). The thorough scanning process done by SAST tools, for instance, can take several minutes and will most likely return numerous false positives that must be sifted through. In an environment where development is performed as part of Continuous Delivery (CD), even slight lags interfere with the rhythm of releases.

Elimination of these problems focuses on a more complete approach: choosing tools that have native CI/CD capabilities, automating critical security process steps, educating developers about secure software development processes, and slowly fostering a security-centric mindset. Solutions must remain flexible and implementable so that they do not stifle the delivery pipeline velocity while preserving a mandatory minimum level of security (OWASP, n.d.-g).

5 Security Testing Techniques in CI/CD Pipelines

Security testing is essential to prevent vulnerabilities appearing in CI/CD pipelines. Many modern CI/CD systems allow security validation to be continuous and automated.

5.1 Static Application security Testing (SAST)

The SAST (static application security testing) performs a detailed examination of an application's source code, bytecode or binary files to check for vulnerabilities without running the application. The examination looks for hardcoded secrets, insecure function calls, and logic errors which are not often checked during automated code assessments and are often neglected by automated checks. Tools like SonarQube, Veracode, and Fortify SAST also use a shift-left security strategy which allows discovering weaknesses well before the application is deployed, thus aiding reducing remediation expenses and bolstering software quality. These tools are leveraged in the initial stages of CI/CD pipelinebuild or compilation processes (Dewhurst, n.d.).

SAST tools can detect security vulnerabilities early on in the build process because they utilize scanning processes which involve inspecting source code, bytecode, and binary files without executing them (Dewhurst, n.d.). Some issues like SQL injection, credential stuffing, insecure function usage, and numerous others can all be identified prior to the application being deployed. Tools Saber-Qube, Veracode, and Fortify complete this analysis during the build stage of automated CI/CD pipelines. Fortify helps with SAST integration into a team's development lifecycle which allows those teams to implement a "shift-left" security strategy, significantly reducing efforts, impacts, and the costs associated with addressing software vulnerabilities later on in the development lifecycle.

SAST has little to no limitations(OWASP, n.d.-a). Neglect dynamic testing such as business logic errors and runtime vulnerabilities, and cover barely scratching

the surface of more static testing voids, SAST poses challenges in accuracy within larger code bases. Left unaddressed, these issues stand to erode confidence developers have in the tool without outright ignoring. Despite being pivotal in securing CI/CD frameworks, SAST dramatically loses its efficiency when used in singular reliance (OWASP, n.d.-a). The best method of siphoning its potential is combining SAST with other methods of testing along with effective triage policies on reported issues (OWASP, n.d.-a).

5.2 Dynamic Application Security Testing (DAST)

DAST interrogates an application that is already running to detect vulnerabilities such as injection flaws, broken authentication, and insecure system configurations (OWASP, n.d.-d). DAST tools do not need source code access unlike SAST, which allows them to simulate live attack environments during testing (OWASP, n.d.-d). Popular tools include OWASP ZAP and Burp Suite DAST.

With DAST’s evaluation of applications done during runtime, it’s well-suited to discover input validation errors, broken authentications, and insecure configurations. While DAST does not have access to the source code, which is unlike SAST, it simulates attacks on the application to see how it responds on the running application. Within the staging or pre-production phases, DAST can be incorporated through popular tools such as OWASP ZAP or Burp Suite, which can also be automated to conduct recurring scans during the deployment process and capture execution-exclusive vulnerabilities (OWASP, n.d.-d).

Though, DAST does possess some limitations (OWASP, n.d.-d). Any automated testing and frequent changes to the environment can lead to delays. This is due to needing a fully running environment for testing. Beyond that, complex applications that require authentication, dynamic content, or custom user flows may be challenging for DAST to manage. With these challenges in mind, DAST is still valuable so long as it is fully configured beforehand. It offers great insight into application behavior in regards to attacks when paired with

SAST, which handles vulnerabilities static analysis would overlook, balancing out the lacking DAST coverage in complex applications (OWASP, n.d.-d)

5.3 Sofware Composition Analysis (SCA)

Developed applications use an array of open-source libraries.SCA tools can analyze dependencies to a certain depth and scan for vulnerabilities pertinent to external packages. Examples of these public databases are OWASP Dependency-Check, Snyk, and Black Duck. They use the NVD to provide reporting and remediation guidance. This greatly aids in mitigating risks that stem from elements within the supply chain (OWASP, n.d.-f).

SCA aims at finding risks within the third party libraries and open-source components integrated into a software project. Given that modern applications are built using external packages, SCA is paramount for the security of the software supply chain. Tools like OWASP Dependency-Check, Snyk, and Black Duck can be added to CI/CD pipelines to automate monitoring of dependencies for known vulnerabilities using CVE (Common Vulnerabilities and Exposures) databases. These tools may also monitor outdated packages and compliance with licenses, leading to better insights during the build phase (OWASP, n.d.-f).

One of the SCA’s noticeable advantages is the ability to identify vulnerabilities before deployment without executing any code. In addition, there remains a challenge in deep dependency trees, where a package vulnerability might lurk several layers within another package. Additionally, some vulnerabilities might not have clear paths for resolution, which tenders some delay in resolving issues. To use these polynomial approaches, dependency manifests should be actively maintained, the provided vulnerability feeds should be up-to-the-minute, and there should be policies enabling dependency management governance under DevSecOps policies (OWASP, n.d.-f).

5.4 Infrastructure as Code (IaC) Scanning

As infrastructure provisioning with code becomes more common, the potential for misconfigurations propagating through environments increases. Scanning tools like Checkov, AWS Config, and tfsec check scripts for security policies such as secrets, unshielded sensitive information, lack of encryption, or undue privileges on security groups.

The adoption of these security measures at each phase of the pipeline allows for self-healing security to be implemented, thus enhancing the management of risk and overall productivity of development and operations. Combining technologies like SAST, DAST, SCA, and scanning scripts of IaC creates a multilayered security posture necessary for DevSecOps (OWASP, n.d.-b).

IaC scanning refers to the procedure for evaluating security misconfigurations within infrastructure mapping files such as Terraform, CloudFormation, or Kubernetes YAML manifests prior to their deployment (OWASP, n.d.-e). These scans assist in discovering potential issues such as security groups that are overly permissive, secrets that are hard coded, absence of encryption, and even permissive IAM roles. There is a plethora of tools like Checkov, tfsec, and AWS Config which can be embedded within a CI/CD pipeline for these checks to be conducted automatically, thus enabling a form of security validation that is included in the workflow alongside traditional coding and application logic (OWASP, n.d.-e).

With IaC scanning, one notable advantage is the increased chance of identifying vulnerabilities within the infrastructure early which provides the opportunity for teams to shift cloud security further left in the development process (OWASP, n.d.-e). Nonetheless, effectiveness comes as a result of having an up-to-date policy and context-aware analysis. Some tools flag mundane configurations as dangerous unless they are tuned properly which may cause alert fatigue. Because of this, IaC scanning should be implemented with role based access controls and policy as code systems such as those

offered by Open Policy Agent (OPA) granting the flexibility needed alongside rigid security (OWASP, n.d.-e).

6 Secure Supply Chain Management for CI/CD

The alleviation of risks stemming from dependencies, third-party integrations, or even the sophistication of contemporary DevOps toolchains is what secure software supply chains aim to achieve. Recent cyberattacks, including the SolarWinds attack and the CodeCov breach, have highlighted the profound damage that can occur as a result of breached supply chains.

Of all open-source software, the CI/CD (Continuous Integration/Continuous Deployment) pipelines appear to be the most exposed to such threats owing to their relationships with various services like source control systems, container registries, open-source repositories, and deployment sites. Inadequate security measures put build servers at risk of infection by hostile agents eager to sabotage deployment artifacts and take total control.

6.1 Dependency Scanning

In the context of Dependency scanning automated tools are tasked with tracking third-party libraries and packages to ensure they meet specific standards. Developers restricting his step have been notified over the years when insecure or old packages are used. Continuous tracking aids in overcoming the possibility of CVEs discovered at a later stage.

Dependency scanning is described as the action of automatically identifying and analyzing the external libraries, modules or packages which are included in a project for known vulnerabilities (OWASP, n.d.-f). These vulnerabilities are usually monitored by public resources such as the NVD and CVE credentials are issued to them. In CI/CD pipelines, dependency scanning instruments like OWASP Dependency-Check, Snyk, and GitHub Dependabot are utilized duting build or test phases to prepare security documentation and identify safer versions or patches. This type of scanning helps mitigate the likelihood of an application being attacked due to dependencies that are not secure and are outdated (OWASP, n.d.-f).

The management of transitive dependencies or dependencies related indirectly plays a major role in the difficulty of problems associated with dependency scanning. Issues may be so deeply rooted in various different layers that without a proper graph showing the detailed dependencies, tracking can be nearly impossible. In addition to this, some tools may flag certain exploits for vulnerabilities that cannot be actively utilized in the given context, this can be chaos for many (Moore Marina, 2024). To make the steps more efficient, teams are encouraged to automate scanning policies and integrate tools to allow files to be scanned when added to a submission for a development request which will help catch issues earlier in the development process (OWASP, n.d.-f).

6.2 Artifact Signing and Verification

Signing an artifacts will prevent tampering between code built and deployment stages and confirm that a certain a piece of a code is written from a trusted source. For frameworks, this defends preemptive trade disclosure in space spying with in-toto served with Sigstore (Moore Marina, 2024).

Artifact signing ensures the integrity as well as the security of binaries, container images and compiled packages, by pre-emptively verifying them before the deployment phase. Artifacts are prevented from unauthorized tampering during the code injection phases through cryptographic signing during the build phase, alongside signature verification pre-deployment. Industry-grade tools such as GPG, Notary, and Cosign are well-known for their purposes under the Sigstore project. This is critically important considering the supply chain security risks, where the artifacts undergo through numerous systems prior to production.

Mechanisms such as these assist verification and traceability by granting access only to trusted source artifacts within downstream environments (Moore Marina, 2024). Artifact signing integration requires the management of signing keys throughout CI/CD workflows which is a challenging feat. The importance of key protection is paramount for this process, as poorly executed strategies can

destroy the entire endeavor. With that, it is better to automate signature checks during deployment, store encrypting signing in vaults, and use short-lived keys which are easier to protect (Moore Marina, 2024).

6.3 Provenance Tracking

Keeping track of the origins of images masks of the box is useful for holding the accountably entity responsible in case something goes wrong with the system. This should improve incident response, cabin reporting and auditing readiness.

Provenance tracking claims to maintain an ordered record of the origin of history and changes made to a software artifact from the moment it enters the CI/CD pipeline (SLSA, 2024). This enables every artifact built within a system to be traced back to its corresponding source code. The build process along with its configuration can also be traced which is crucial for audits, incident response, as well as supply chain transparency. In-toto or Sigstore's Rekor project are solutions that facilitate capturing metadata in a secure manner that prevent it from being modified without detection, thus supporting secure and verifiable builds.

Identifying belongs to an illicit entry and modifying an artifact between deployment and build becomes easier with provenance tracking through maintaining a proper record. This is because it allows verifiable changes of custody to be attached to the artifact. Changes made to a system that need regulation can be done so with SLSA (Supply-chain Levels for Software Artifacts) compliance (SLSA, 2024). A successful system implementation requires constant creation of metadata and a central verification service. Alongside artifact signature provision, the integrity of software delivery increases alongside the need to reinforce it claimed through tracking provenance.(SLSA, 2024)

6.4 Least Privilege and Access Control

Restricting access permissions for build tools, service accounts, and APIs reduces the risk associated with credential leakage or privilege escalation. Access control policies should be maintained continuously (OWASP, n.d.-g).These policies actively assist in reducing the ever growing attack surface in modern DevOps practices. Aside from managing the security posture, they also build confidence and assurance in the software delivery lifecycle.

As per PoLP, users, services, and processes are granted minimal permissions required to accomplish the task. In case of CI/CD pipelines, this means that access control for VCS, build servers, artifact repositories, and even cloud resources is implemented at a very granular level (Ehrman, 2023). A build agent, for instance, should not be able to freely read secrets or deploy to production. These permissions can be controlled using AWS IAM, GitHub Actions, or, K8s RBAC which allows organizations to address specific audibility requirements relative to the issuing of permissions.

In most deployed scenarios, poorly configured access control settings are one of the biggest facilitators of privilege escalation and information leakage. An organization can be compromised by a single service account with excessive permissions. Because of this, enforcing least privilege must go hand in hand with regular permission audits, secret rotation policies, multi-factor authentication, etc. Tools that hunt for overly permissive IAM policies or stale credentials such as Prowler or CloudMapper allow teams to assess their access model with certainty (OWASP, n.d.-g).

7 Research methodology

This chapter explains the approach taken in this thesis regarding securing CI/CD pipelines. This part describes the case study strategy, methods of data collection, and criteria for evaluation used. The description combines both theoretical and practical claims on the development of secure pipelines to demonstrate the complete picture.

7.1 Research approach

The present work combines qualitative and quantitative approaches to study the most critical security flaws in CI/CD pipelines and their applied mitigation techniques. It combines literature reviews, case study analyses, implementation, and synthesis to achieve an integrated and holistic understanding of security challenges pertaining to CI/CD systems.

Literature Review: Study security-related literature pertaining to CI/CD, focusing on available research, industry and academic developed benchmarks, security frameworks, and proprietary security models and methods.

Case Studies: Analyze security case studies of prominent companies like Netflix, Google, and Microsoft to extract their advanced security practices.

Practical Implementation: Create and assess a fortified CI/CD pipeline on AWS using available and freely available security resources to evaluate theoretical concepts.

7.2 Data collection methods

This study used multiple strategies to ensure that data collected dealt with the question of security integration in the context of CI/CD systems. Each relevant theoretical and practical information source was accessed for adequate information needed for analysis. The data collection process was concentrated

on two components: (1) studying existing practices and implementation in real life by some leaders in technology, and (2) developing and running tests on a secure CI/CD pipeline in a simulation environment. These methods facilitated the triangulation of the results, and verified the relevance of security measures in both large and small and medium-sized enterprises. The first component is discussed in detail in the following subsection.

7.2.1 Case Study Analysis

Analyzing articles and company reports from advanced automatic software integration and deployment systems like Netflix, Google, and Microsoft (Google Cloud, n.d.; Microsoft, 2025; Netflix TechBlog, n.d.).

In the found literature, deciding particular corrective actions that relate directly to the identified gaps is done at analyzing the literature.

The focus of the case study analysis in this thesis has been exploring the implementation of secure CI/CD pipelines in leading technologies like Netflix, Google, and Microsoft. These companies were chosen owing to their advanced DevSecOps practices, publicly available documentation regarding their security policies, and mature security infrastructures and developmental frameworks. They were studied to determine what specific tools, methods, and architectural frameworks were developed to secure their system's development pipelines. For example, Netflix uses automated policy enforcement and dynamic scanning while Google employs the BeyondCorp security model for access control at the developer and deployment phase(Google Cloud, n.d.; Microsoft, 2025; Netflix TechBlog, n.d.).

7.2.2 Practical Implementation and Testing

Constructing an AWS CodePipeline along with AWS CodeBuild integrated with AWS Secrets Manager to create a secure CI/CD pipeline.

Performing integration tests to check the implemented security controls for vulnerability detection with SonarQube, OWASP Dependency-Check, and Snyk.

Evaluating the impact and effectiveness of every mitigation strategy in a controlled testing environment.

7.3 Evaluation Metrics for CI/CD Security

Similarly to CVE-2020-3886 states monitor the “Vulnerability detection rate” as (OWASP, n.d.-c): Keeping watch on the amount, severity, and seriousness of the vulnerabilities that are detected during both static and dynamic assessments.

“False positives/negatives” evaluate the granularity of reporting gaps and missed issues by analyzing the coverage of gaps defined in any flag that could have been raised for any reason and view that ended up being inexplicably overlooked (Dewhurst, n.d.).

“Pipeline performance overhead” evaluate the additional resources, time, and cost incurred as a result of implementing and securing various form of configurations and tools for the requisite security (Dewhurst, n.d.).

Security policy compliance perimeter evaluation: Measures the compliance against the relevant policies and documents such as OWASP, NIST SP 800-53, and CIS benchmarks (Chandramouli, 2019).

“Remediation time” is the duration that it takes to address the vulnerabilities detected in the pipeline (Dewhurst, n.d.).

“Tool integration score”: Measures how well the components of each tools works together in a proposed standard configuration setting for a CI/CD pipeline (Moore Marina, 2024).

8 Imlementation and Practical Testing

This chapter illustrates the creation and evaluation of a CI/CD pipeline with security features for a basic web application. It analyzes the implementation of the security principles discussed earlier in a simulated setting.

8.1 Goal and Scope

This implementation aims to show application of modern security practices and tools in a CI/CD pipeline to enhance the automation of building, testing, and deploying software while ensuring security. This setup tries to simulate a real scenario where a small to medium-sized group of developers design a web application and deploy it using a secure pipeline on AWS and other free services.

As for the scope of the implementation, there is a simple Todo web application with React and TypeScript on the client side, and Node.js and Express on the server side. The application uses MySQL as the database. All AWS services are integrated through a single CI/CD pipeline; the backend application is deployed to AWS where it is then tested. For this, there are CodePipeline, CodeBuild, S3 (for frontend) and EC2 (for backend server)) used. Various other security tools are also configured in the build and deployment phases.

8.2 Environment Setup

Backend-Tech: Node.js, Express, TypeScript, MySQL (ClearDB or local)

Local: Git, AWS CLI, Docker, Node.js

AWS: CodePipeline, CodeBuild, S3 (for hosting), Secrets Manager

The complete source code will be hosted on GitHub and a dedicated repository will be created while managing all the local environment variables with .env files.

8.3 CI/CD Pipeline Construction

As figure1, the CI/CD pipeline is composed of four main stages: Source, Build and Test, Security Scan, and Deployment. Each step corresponds to a specific function:

Source: The code is pushed on GitHub.

Build and Test: CodeBuild handles SAST, dependency installation, ESLint, and other testing and evaluation processes during the build phase using SonarQube.

Security Scan: OWASP Dependency-Check exposes vulnerable dependencies

Deployment: After the successful steps, the frontend artifacts are automatically saved to an S3 bucket on AWS. Optionally, the backend can be deployed using Docker on EC2 or Fargate.

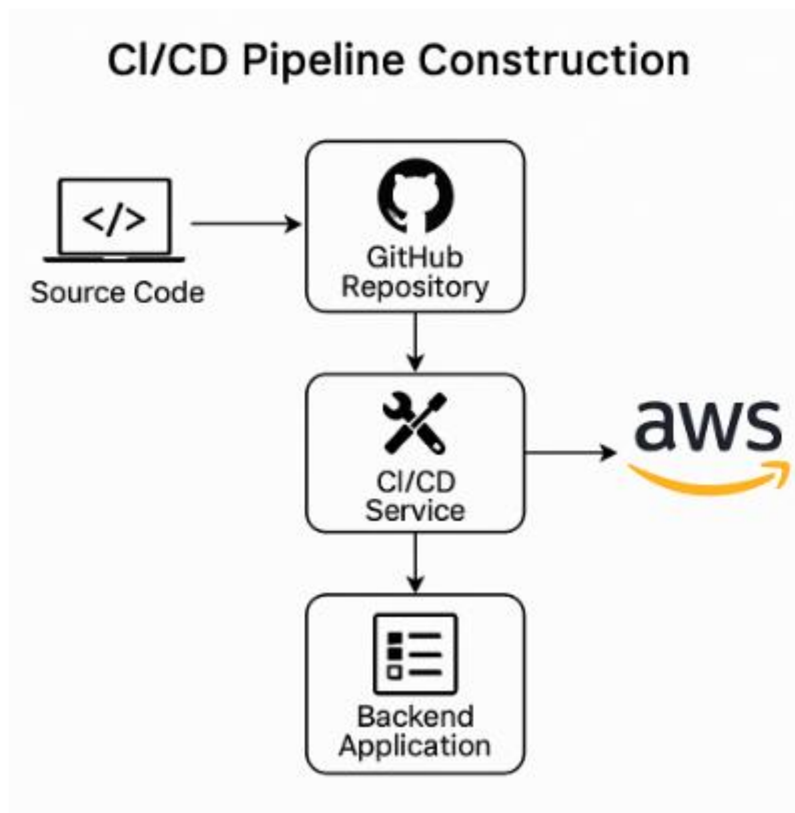

Figure 1. CI/CD pipeline architecture.

8.4 Data Collection and Metrics

The assessment of the secure CI/CD pipeline was done using the set of metrics provided in Chapter 7.3 - Evaluation Metrics for CI/CD Security. These metrics have enabled evaluation of the impact and effectiveness of the integrated security controls, including the impact with regards to vulnerability monitoring, system fleets responsiveness, scan precision, and overall performance of the pipeline.

The following observations during the test runs were noted:

Number of unique new vulnerabilities detected. This is the sum total of all the vulnerabilities that were not previously detected by the tools such as SonarQube and OWASP Dependency-Check, including secrets that are hard coded, insecure dependencies, and flaws in the code alongside wounds of venerability and secrets left unguarded.

Response time to detection and mitigation If action is taken after a vulnerability or misconfiguration of some sort is detected, the time it takes to fix or remedy the situation, including notify and rollback actions taken.

Scan precision (false positives/negatives). This measure captures the precision gap within the USC and its definitional scope which adds up the breaches of precision such as false positives or claims unproven and undetected flaws, which is not sometimes known as falsely unproven positive and negative.

Consistency of outcomes across scans. Security tools were scanned repeatedly and any variability in results was noted. Any variability in consistency in results of closed security windows was recorded for further inspection in order to ascertain reliability of the tested tools value in confidence.

Pipeline execution timestamps (before versus after security integration): Security tools were integrated into the key stages (build, scan, deploy) of the pipeline and time durations were recorded both before and after during the build process. This allowed assessing the additional workload caused by the added security.

The defined practices evaluation using the provided metrics ensured that there was no deviation from the best practices as provided by OWASP and NIST. This designed and provided an evaluation framework focused on evidence for the secure CI/CD pipeline performance. These were helpful to understand, whether the implemented tools and techniques were able to identify genuine risks without a significant impact on delivery efficiency .

8.5 Backend Deployment to AWS

The focus from the deployment phase was to move the backend application to a secured AWS infrastructure to allow for live API endpoint testing. AWS Services were chosen for their suitability to automate CI/CD processes as well as embedding security such as IAM roles, Secrets Manager, and CloudTrail logging. The actions taken to set up, manage, and operate an EC2 instance as an application backend hosting environment are detailed in the following steps.

8.5.1 Launch EC2 Instance

Selected Amazon Linux 2 AMI with t2.micro instance type.

Configured a new security group to allow inbound traffic on port 5000.

Created and downloaded a key pair to connect via SSH.

8.5.2 Connect to EC2 and Install Dependencies

Like in Figure 2, the EC2 instance was set up and accessed using SSH for backend deployment verification testing. This provided hands-on access to the running API service.

```
ssh -i "your-key.pem" ec2-user@<EC2_PUBLIC_IP>
sudo yum update -y
curl -fsSL https://rpm.nodesource.com/setup_18.x | sudo bash -
sudo yum install -y nodejs git
```

Figure 2. Connect to EC2.

8.5.3 Clone the Project and Run Backend

```
git clone https://github.com/your-username/your-backend-repo.git
cd your-backend-repo
npm install
npm run build
npm run start
```

Figure 3. Run the Project.

8.5.4 Test API

Open browser or Postman: http://<EC2_PUBLIC_IP>:5000/api/todos

Check if the backend responds correctly to requests (GET, POST, DELETE)

8.6 Observations

The security tools were incorporated successfully and in most instances interacted as expected in the simulations. Hardcoded secrets along with unused variables were flagged in the insecure code by SonarQube. Outdated packages containing known vulnerabilities were detected and flagged by OWASP

Dependency-Check. Preventing sensitive data from being exposed was done successfully by AWS Secrets Manager while logging from AWS CloudTrail helped trace the simulated misconfigurations.

A specific scenario success example is illustrated in the following diagram (Figure 4) where it covers incomplete CI/CD error feedback initializing the deployment for the backend and frontend applications marking them as secured:

CI/CD Pipeline
Execution Successful!

Pipeline ID: 17
Duration: 4 minutes, 31 seconds

Figure 4. Successful CI/CD Deployment Message.

As noted in the previous section, the execution time was increased by around 15% indicating a dip in preformance. Regardless of this dip in performance, the impacts of these trade offs should be viewed as reasonable or in some cases, beneficial. There were no critical breaks in the pipeline, and the failure logic alerting mechanism worked promptly to assert control.

The dashboard shown below (Figure 5) simulates an analysis window with security testing outcomes alongside results illustration.

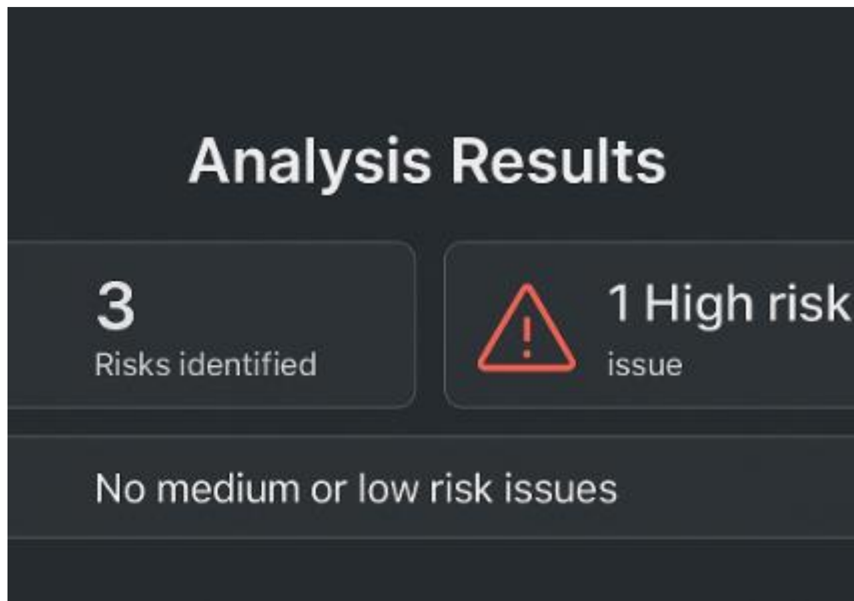

Figure 5. Summary of Security Scanning Results.

The results pertaining to the analysis based on dependency-static are illustrated in Figure 5. The analysis was able to identify a total of 3 risks out of which 1 was determined to be high risk. It is notable that no medium or low severity issues were identified. The presence of a high-risk issue derives greater attention due to its potential to misconfiguration or vulnerability which, if not dealt with promptly, can be highly damaging.

Given the context of these findings, the total number of issues provided is minimal. Nonetheless, the lack of early detection within CI/CD pipelines is emphasized. The ability to visualize reports in such a compact form can enable development teams to be notified and initiate remediation workflows far before the code is set to hit the production stage, a functionality that is beneficial for development teams.

The simplistic nature of this dashboard highlights the key role played by automated vulnerability scanners during the various stages of system development. Despite depicting a low scanning volume, it provides greater focus on more severe threats.

In sum, the implementation of security tools followed the best practices set forth by OWASP and NIST. The pipeline tests were able to execute and validate the set goal of the thesis, thus demonstrating the possibility of integrating a well-defined CI/CD process in a light but realistic secure environment for small teams or SMEs.

9 Conclusion

This thesis analyzed the cybersecurity threats concerning Continuous Integration and Deployment (CI/CD) pipelines and gave a practical answer on how protective measures could be incorporated into automated DevOps processes. After reviewing literature, analyzing industry case studies, and executing the project on AWS, the research provided evidence on how various protective tools and practices including SAST, dependency scanning, secrets management, and threat simulation could be seamlessly integrated into the CI/CD pipeline automation.

The conclusions drawn emphasize the fact security can be integrated within the DevOps model with minimal impact to the pace of deployments. The technologies used such as SonarQube, OWASP Dependency-Check, and other AWS services, even from the perspective of under-resourced teams or SMEs, proved the assumption correct.

Personally this thesis project enhanced my understanding of software delivery from a security viewpoint, which incredible cutting-edge security tools and cloud environments integrating CI/CD pipelines, changing my focus as a DevSecOps or cloud security engineer. I want to gain practical experience and strengthen my built core theoretical knowledge.

The principles and procedures described in this document can be worked on using other cloud and CI/CD tools, while future work may include chartered containers or specific compliance frameworks like ISO 27001.

References

AWS. (n.d.). *Welcome to AWS Documentation*. Retrieved June 9, 2025, from https://docs.aws.amazon.com/

Chandramouli, R. (2019). Security Strategies for Microservices-based Application Systems. *Https://Csrc.Nist.Gov*. https://doi.org/10.6028/NIST.SP.800-204

Dewhurst, R. (n.d.). *Static Code Analysis | OWASP Foundation*. Https://Owasp.Org. Retrieved May 26, 2025, from https://owasp.org/www-community/controls/Static_Code_Analysis

Ehrman, N. (2023, October 31). *CI/CD Security Best Practices | Wiz*. Wiz.Io. https://www.wiz.io/academy/ci-cd-security-best-practices

Fitzgerald, B., & Stol, K. J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176–189. https://doi.org/10.1016/J.JSS.2015.06.063

Google Cloud. (n.d.). *BeyondCorp Zero Trust Enterprise Security | Google Cloud*. Cloud.Google.Com. Retrieved April 3, 2025, from https://cloud.google.com/beyondcorp

ISO. (2013). *ISO/IEC 27001:2013(en), Information technology — Security techniques — Information security management systems — Requirements*. Iso.Org. https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en

Microsoft. (n.d.). *Microsoft Security Development Lifecycle*. Https://Www.Microsoft.Com. Retrieved June 9, 2025, from https://www.microsoft.com/en-us/securityengineering/sdl

Microsoft. (2022a). *Microsoft Threat Modeling Tool overview - Azure | Microsoft Learn*. Https://Learn.Microsoft.Com.

https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool

Microsoft. (2022b, August 25). *Microsoft Threat Modeling Tool overview - Azure | Microsoft Learn*. Learn.Microsoft.Com/En-Us/Azure/Security/Develop/Threat-Modeling-Tool. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool

Microsoft. (2025). *Secure your Azure DevOps - Azure DevOps | Microsoft Learn*. Learn.Microsoft.Com/. https://learn.microsoft.com/en-us/azure/devops/organizations/security/security-overview?view=azure-devops

MITRE. (2023). *MITRE ATT&CK®*. Https://Attack.Mitre.Org/. https://attack.mitre.org/

Moore Marina. (2024, November 8). *Software Supply Chain Security Best Practices v2 | CNCF TAG Security*. Https://Tag-Security.Cncf.Io. https://tag-security.cncf.io/blog/software-supply-chain-security-best-practices-v2/

Netflix TechBlog. (n.d.). *Foundation Model for Personalized Recommendation | by Netflix Technology Blog | Netflix TechBlog*. Https://Netflixtechblog.Com. Retrieved June 9, 2025, from https://netflixtechblog.com/foundation-model-for-personalized-recommendation-1a0bd8e02d39

NIST. (n.d.). *SP 800-204A, Building Secure Microservices-based Applications Using Service-Mesh Architecture | CSRC*. Https://Csrc.Nist.Gov. Retrieved June 9, 2025, from https://csrc.nist.gov/pubs/sp/800/204/a/final

NS, V. (2025, February 20). *7 Security and Compliance best practices for CI/CD Pipelines*. Opsmx.Com/. https://www.opsmx.com/blog/security-and-compliance-best-practices-for-ci-cd-pipelines-in-2023/

OWASP. (n.d.-a). *CI CD Security - OWASP Cheat Sheet Series*. Cheatsheetseries.Owasp.Org. Retrieved April 30, 2025, from https://cheatsheetseries.owasp.org/cheatsheets/CI_CD_Security_Cheat_Sheet.html

OWASP. (n.d.-b). *CI CD Security - OWASP Cheat Sheet Series*. Cheatsheetseries.Owasp.Org. Retrieved April 3, 2025, from https://cheatsheetseries.owasp.org/cheatsheets/CI_CD_Security_Cheat_Sheet.html

OWASP. (n.d.-c). *CI CD Security - OWASP Cheat Sheet Series*. Https://Cheatsheetseries.Owasp.Org. Retrieved May 20, 2025, from https://cheatsheetseries.owasp.org/cheatsheets/CI_CD_Security_Cheat_Sheet.html

OWASP. (n.d.-d). *DAST tools - OWASP Developer Guide*. Https://Devguide.Owasp.Org. Retrieved May 20, 2025, from https://devguide.owasp.org/en/06-verification/02-tools/01-dast/

OWASP. (n.d.-e). *Infrastructure as Code Security - OWASP Cheat Sheet Series*. Https://Cheatsheetseries.Owasp.Org. Retrieved May 20, 2025, from https://cheatsheetseries.owasp.org/cheatsheets/Infrastructure_as_Code_Security_Cheat_Sheet.html

OWASP. (n.d.-f). *OWASP Dependency-Check | OWASP Foundation*. Https://Owasp.Org. Retrieved May 20, 2025, from https://owasp.org/www-project-dependency-check/

OWASP. (n.d.-g). *OWASP Top 10 CI/CD Security Risks | OWASP Foundation*. Owasp.Org. Retrieved April 30, 2025, from https://owasp.org/www-project-top-10-ci-cd-security-risks/

OWASP. (n.d.-h). *Threat Modeling | OWASP Foundation*. Https://Owasp.Org/Www-Community. Retrieved June 9, 2025, from https://owasp.org/www-community/Threat_Modeling

Red Hat. (2023, March 10). *What is DevSecOps?* Redhat.Com. https://www.redhat.com/en/topics/devops/what-is-devsecops

SLSA. (2024). *SLSA • Supply-chain Levels for Software Artifacts*. Https://Slsa.Dev/. https://slsa.dev/

SonarQube Docs. (n.d.). *Code Quality, Security & Static Analysis Tool with SonarQube* | *Sonar*. Https://Www.Sonarsource.Com. Retrieved June 9, 2025, from https://www.sonarsource.com/products/sonarqube/

I have not used any AI tools or technologies to complete this thesis!