



University of Piraeus  
School of Information and Communication Technologies  
Department of Digital Systems

Undergraduate Program of Study/Postgraduate Program of Studies  
MSc Digital Systems Security

Master Thesis

Enhancing Application Security through DevSecOps: A Comprehensive  
Study on Vulnerability Detection and Management in Continuous  
Integration and Continuous Delivery Pipelines

Supervisor Professor: Stefanos Gritzalis

Name-Surname	E-mail	Student ID.
Pagona Vourou	<a href="mailto:peggy.vourou@ssl-unipi.gr">peggy.vourou@ssl-unipi.gr</a>	mte2102

Piraeus  
27/10/2023



## Περίληψη

Αυτή η διπλωματική εξετάζει τον συναντισμό των μεθόδων ανάπτυξης λογισμικού και της κυβερνοασφάλειας στον τομέα του DevSecOps, ένα παράδειγμα που δίνει προτεραιότητα στην ενσωμάτωση μέτρων ασφαλείας σε όλον τον Κύκλο Ζωής Ανάπτυξης Λογισμικού (SDLC). Αυτό το έγγραφο εξετάζει τις δυσκολίες που σχετίζονται με την εφαρμογή του DevSecOps, συμπεριλαμβανομένης της ανίχνευσης και της επίλυσης των ευπαθειών, καθώς και της συνεχούς παρακολούθησης και της αντιμετώπισης των απειλών ασφαλείας εντός των σύγχρονων αγωγών ανάπτυξης λογισμικού.

Η αρχή του ταξιδιού περιλαμβάνει μια πρώτη εξερεύνηση του Κύκλου Ζωής Ανάπτυξης Λογισμικού (SDLC) και μια σφαιρική ανάλυση των βασικών εννοιών που αναφέρονται στο Agile Manifesto, τα οποία αποτελούν τη βάση για τις σύγχρονες πρακτικές ανάπτυξης λογισμικού.

Αυτή η διατριβή επικεντρώνεται στο Application Security Pipeline για την Συνεχή Ενσωμάτωση/Συνεχή Παράδοση (CI/CD), εξετάζοντας τον συναντισμό της Πληροφοριακής Ασφάλειας και της Ασφάλειας Εφαρμογής. Υπογραμμίζεται η σημαντικότητα της ασφάλειας στο πλαίσιο των διαδικασιών DevOps καθώς και οι δυσκολίες που αντιμετωπίζουν οι επιχειρήσεις κατά την εφαρμογή των αρχών του DevSecOps.

Επιπρόσθετα, προσφέρει μια λεπτομερή ανάλυση των ευπαθειών, επικεντρώνοντας ειδικά στις ευπαθείς της εφαρμογής, τις ευπαθείς των εικόνων Docker και τις ευπαθείς μέσα στο Pipeline CI/CD, εξετάζοντας διάφορες προσεγγίσεις ανίχνευσης ευπαθειών κατά μήκος του Κύκλου Ζωής Ανάπτυξης Λογισμικού (SDLC), συμπεριλαμβανομένης της Στατικής Ανάλυσης Κώδικα, της Δυναμικής Ανάλυσης Κώδικα, της Ανάλυσης Σύνθεσης Λογισμικού και της Ανάλυσης Ασφάλειας Εφαρμογής.

Η κύρια συνεισφορά αυτής της μελέτης βρίσκεται στην πρόταση και αξιολόγηση ενός παραδείγματος Συνεχούς Διαχείρισης Ευπαθειών εντός του πλαισίου του DevSecOps. Το έγγραφο αναλύει τις πρακτικές της Συνεχούς Αξιολόγησης, Αντιμετώπισης και Αναφοράς Ευπαθειών, υπογραμμίζοντας τη σημασία της υιοθέτησης μιας προακτινόμενης και επαναληπτικής μεθοδολογίας για τη διαχείριση των ευπαθειών.

Για να επικυρώσει το προτεινόμενο πλαίσιο, παρουσιάζεται μια πρακτική μελέτη περίπτωσης χρησιμοποιώντας την εφαρμογή OWASP WebGoat. Αναλύεται η αποτελεσματικότητα διαφόρων εργαλείων ασφαλείας, συμπεριλαμβανομένων της SAST με το SNYK, της SAST με το SonarQube, της Ανάλυσης Ασφαλείας των Εικόνων Docker με το Trivy και το Grype, καθώς και της DAST με το OWASP-Zap και το Arachni. Ο στόχος είναι να αξιολογηθεί η αποτελεσματικότητα αυτών των εργαλείων και να παρασχεθούν σημαντικές προτάσεις.

Ο στόχος αυτής της διπλωματικής είναι να δημιουργήσει μια σύνδεση μεταξύ της ανάπτυξης λογισμικού και της κυβερνοασφάλειας, παρέχοντας χρήσιμες γνώσεις και πρακτικές συμβουλές σε επιχειρήσεις που επιδιώκουν να ενισχύσουν τα μέτρα

ασφαλείας τους στο συνεχώς μεταβαλλόμενο πεδίο της σύγχρονης ανάπτυξης λογισμικού.

## Abstract

This thesis examines the intersection of software development methods and cybersecurity in the domain of DevSecOps, a paradigm that prioritizes the incorporation of security measures across the entire Software Development Lifecycle (SDLC). This paper explores the difficulties associated with implementing DevSecOps, including the detection and resolution of vulnerabilities, as well as the ongoing monitoring and mitigation of security threats within contemporary software development pipelines.

The start of the journey includes a first exploration of the Software Development Lifecycle (SDLC) and a comprehensive analysis of the fundamental concepts outlined in the Agile Manifesto, which serve as the foundation for contemporary software development practices. This paper delves deeper into the examination of Continuous Integration and Continuous Delivery (CI/CD) processes, as well as the emergence of DevOps as a disruptive influence inside the industry.

This paper focuses on the Application Security Pipeline for Continuous Integration/Continuous Deployment (CI/CD), examining the intersection of Information Security and Application Security. This statement highlights the significant importance of security in the context of DevOps processes, as well as the difficulties that businesses have while implementing DevSecOps concepts. Also, it provides a thorough examination of vulnerabilities, specifically emphasizing application vulnerabilities, Docker image vulnerabilities, and vulnerabilities inside the CI/CD pipeline, and examines different ways for detecting vulnerabilities across the Software Development Life Cycle (SDLC). These approaches include Static Code Analysis, Dynamic Code Analysis, Software Composition Analysis, and Container Security Analysis.

The primary contribution of this study is to the proposal and assessment of a Continuous Vulnerability Management paradigm within the context of DevSecOps. The document delineates the practices of Continuous Vulnerability Evaluation, Treatment, and Reporting, underscoring the significance of adopting a proactive and iterative methodology towards vulnerability management.

In order to authenticate the suggested framework, a practical case study is shown utilizing the OWASP WebGoat application. This study does a comparative examination of several security technologies utilized in a DevSecOps pipeline. The tools examined include SAST with SNYK, SAST with SonarQube, Container Security with Trivy and Grype, as well as DAST with OWASP-Zap and Arachni. The objective is to evaluate the efficiency of these tools and get valuable insights.

The objective of this thesis is to establish a connection between software development and cybersecurity, providing useful insights and practical advice for businesses seeking to enhance their security measures in the ever-changing realm of contemporary software development.

## Table of Contents

<b>Introduction</b>	<b>2</b>
<b>Thesis Topic</b>	<b>2</b>
<b>Software Development Lifecycle (SDLC)</b>	<b>4</b>
<b>Agile Manifesto</b>	<b>4</b>
<b>Continuous Integration and Continuous Delivery</b>	<b>5</b>
<b>DevOps</b>	<b>6</b>
<b>Pipeline</b>	<b>7</b>
<b>Application Security Pipeline for CI/CD</b>	<b>8</b>
<b>Information Security</b>	<b>8</b>
<b>Application Security</b>	<b>8</b>
<b>Security meets DevOps</b>	<b>9</b>
Challenges on adopting DevSecOps	10
<b>Vulnerabilities</b>	<b>11</b>
Definition	11
Application Vulnerabilities	11
Docker Image Vulnerabilities	17
Pipeline Vulnerabilities	18
<b>Vulnerability detection approaches throughout the SDLC</b>	<b>20</b>
Testing Types	20
Static Code Analysis	21
Dynamic Code Analysis	22
Software Composition Analysis	23
Container Security Analysis	24
<b>Continuous Vulnerability Management in DevSecOps</b>	<b>24</b>
Continuous Vulnerability Evaluation	25
Continuous Vulnerability Treatment	26
Continuous Vulnerability Reporting	27
<b>Continuous Vulnerability Finding in DevSecOps – Case Study: OWASP WebGoat</b>	
<b>Application / Tools Compare</b>	<b>29</b>
<b>Approach and all Set-up</b>	<b>29</b>
Approach	29
Set Up	31
<b>Jenkins</b>	<b>31</b>
<b>OWASP WebGoat</b>	<b>34</b>
<b>SAST using SNYK</b>	<b>36</b>
<b>SAST using SonarQube</b>	<b>44</b>
<b>Container Security using Trivy</b>	<b>52</b>
<b>Container Security using Grype</b>	<b>57</b>
<b>DAST Security using OWASP-Zap</b>	<b>64</b>
<b>DAST Security using Arachni</b>	<b>70</b>

<b>Tools Comparison</b>	<b>78</b>
<b><i>Conclusion</i></b>	<b>83</b>
<b><i>Bibliography</i></b>	<b>85</b>

## Table of Figures

<i>Figure 1: DevOps</i>	6
<i>Figure 2: CI/CD</i>	7
<i>Figure 3: DevSecOps Jenkins pipeline</i>	29
<i>Figure 3: DevSecOps Jenkins approach</i>	30
<i>Figure 4: Jenkins compositionl file</i>	32
<i>Figure 5: Jenkins login screen</i>	33
<i>Figure 6: OWASP WebGoat User Interface</i>	34
<i>Figure 7: OWASP WebGoat Menu</i>	35
<i>Figure 8: OWASP WebGoat cloned via the pipeline</i>	36
<i>Figure 9: SAST SNYK Jenkins pipeline</i>	37
<i>Figure 10: SNYK Plugin</i>	38
<i>Figure 11: SAST SNYK Jenkins pipeline configuration 1</i>	38
<i>Figure 12: SAST SNYK Jenkins pipeline configuration 2</i>	38
<i>Figure 13: SAST SNYK Jenkins pipeline configuration 3</i>	39
<i>Figure 14: SAST SNYK Security Report in Jenkins Pipeline Menu</i>	39
<i>Figure 15: SAST SNYK Security Report Header</i>	40
<i>Figure 16: SAST SNYK Security Report Vulnerabilities</i>	40
<i>Figure 17: SAST SNYK Security Report Vulnerabilities Remediation 1</i>	41
<i>Figure 18: SAST SNYK Security Report Vulnerabilities Remediation 2</i>	41
<i>Figure 19: SAST SonarQube Jenkins pipeline</i>	45
<i>Figure 20: SAST SonarQube Jenkins pipeline configuration</i>	46
<i>Figure 21: SAST SonarQube Security Report in Jenkins Pipeline Menu</i>	47
<i>Figure 22: SAST SonarQube Project Status</i>	47
<i>Figure 23: SAST SonarQube Project Vulnerabilities and Issues of new Code</i>	48
<i>Figure 24: SAST SonarQube Project Vulnerabilities and Issues metrics</i>	48
<i>Figure 25: SAST SonarQube Project Vulnerabilities and Issues of Overall Code</i>	49
<i>Figure 26: SAST SonarQube Project Source Code</i>	49
<i>Figure 27: SCA Trivy Jenkins Pipeline</i>	53
<i>Figure 28: SCA Trivy Jenkins Pipeline configuration</i>	53
<i>Figure 29: SCA Trivy Report in Jenkins Pipeline Menu</i>	54
<i>Figure 30: SCA Trivy Report</i>	55
<i>Figure 31: SCA Grype Jenkins Pipeline</i>	59
<i>Figure 32: SCA Grype Jenkins Pipeline configuration</i>	59
<i>Figure 33: SCA Grype Report in Jenkins Pipeline Menu</i>	60
<i>Figure 34: SCA Grype Report</i>	61
<i>Figure 35: DAST OWASP-Zap Jenkins Pipeline</i>	65
<i>Figure 36: DAST OWASP-Zap execution bash script</i>	65
<i>Figure 37: DAST OWASP-Zap report download</i>	66
<i>Figure 38: DAST OWASP-Zap scanning report</i>	67
<i>Figure 39: OWASP-Zap scanning report results (Summary)</i>	67
<i>Figure 40: DAST OWASP-Zap scanning report results (Alert details)</i>	68
<i>Figure 41: DAST Arachni Jnekins Pipeline</i>	71
<i>Figure 42: DAST Arachni Jnekins Pipeline configuration</i>	72
<i>Figure 43: DAST Arachni execution bash script</i>	72
<i>Figure 44: DAST Arachni report summary</i>	74
<i>Figure 45: DAST Arachni report issues chart</i>	74
<i>Figure 46: DAST Arachni report issues</i>	75
<i>Figure 47: DAST Arachni report plugin results</i>	75
<i>Figure 48: DAST Arachni report sitemap</i>	76
<i>Figure 49: DAST Arachni report configuration</i>	76



## List of Tables

<i>Table 1:Snyk tool results description</i>	42
<i>Table 2:Snyk tool positives and negatives</i>	43
<i>Table 3:SonarQube tool positives and negatives</i>	51
<i>Table 4:Trivy tool positives and negatives</i>	57
<i>Table 5:Grype tool positives and negatives</i>	63
<i>Table 6:OWASP-Zap tool positives and negatives</i>	70
<i>Table 7:Arachni tool positives and negatives</i>	77

# Introduction

## Thesis Topic

The secure creation of software applications holds significant relevance in the current era characterized by fast digital transformation and constantly developing technological environments. The preservation of data and systems' confidentiality, integrity, and availability has emerged as a paramount problem for enterprises on a global scale. The concept of DevSecOps has emerged as a significant reaction to this requirement, promoting the integration of security practices throughout the Software Development Lifecycle (SDLC) in a seamless manner.

This thesis aims to investigate the convergence of software development practices and cybersecurity in the context of DevSecOps. This analysis delves extensively into the problems and possibilities arising from this paradigm change, with a specific emphasis on the identification and resolution of vulnerabilities, as well as the ongoing management of security risks within contemporary software development pipelines.

### Chapter 1: Software Development Lifecycle (SDLC)

Our journey begins with an introduction to the Software Development Lifecycle (SDLC), the foundation of software engineering. The Agile Manifesto, with its principles of flexibility and collaboration, sets the stage for contemporary software development practices. As we delve deeper into the Agile Manifesto, we acquire a better understanding of the guiding values and principles of contemporary software development methodologies.

### Chapter 2: Continuous Integration and Continuous Delivery (CI/CD)

Continuous Integration and Continuous Delivery (CI/CD) are concepts that are introduced by the dynamic nature of software development. These practices, which are intended to expedite the development process, increase productivity, and abbreviate release cycles, hold great promise. However, they present challenges, such as the requirement for comprehensive security measures within the CI/CD pipelines.

### Chapter 3: DevOps

This piece examines the advent of DevOps as a disruptive force in the software development industry. Collaboration, automation, and a culture of shared accountability are promoted by the DevOps principles. In this context, we examine the role of DevOps in bridging the divide between development and operations, along with its security implications.

## Chapter 4: Application Security Pipeline for CI/CD

The development of an Application Security Pipeline for CI/CD environments is a pivotal aspect of our investigation. The convergence of Information Security and Application Security is discussed in this article. We underscore the importance of security within DevOps practices and the difficulties organizations face when attempting to effectively implement DevSecOps concepts.

## Chapter 5: Vulnerabilities

Our thesis focuses heavily on vulnerabilities, a foundational concern in the context of security. We begin by defining vulnerabilities and the potential threats they present. Application vulnerabilities, Docker image vulnerabilities, and CI/CD pipeline vulnerabilities are the focus of our examination.

## Chapter 6: Vulnerability Detection Approaches throughout the SDLC

With vulnerabilities firmly in our sights, we investigate various methods for detecting them throughout the Software Development Lifecycle. Included in this category are Static Code Analysis (SAST), Dynamic Code Analysis (DAST), Software Composition Analysis (SCA), and Container Security Analysis. Within the development pipeline, each methodology has its own strengths and applications.

## Chapter 7: Continuous Vulnerability Management in DevSecOps

The primary contribution of this thesis rests in the proposition and evaluation of a Continuous Vulnerability Management framework within the context of DevSecOps. We describe the Continuous Vulnerability Evaluation, Treatment, and Reporting practices, highlighting the significance of a proactive and iterative approach to vulnerability management.

## Chapter 8: Continuous Vulnerability Finding in DevSecOps – Case Study: OWASP WebGoat Application / Tools Compare

For the purpose of validating our proposed framework, we present a case study using the OWASP WebGoat application. We conduct a comparative analysis of various security tools, including SAST with SNYK, SAST with SonarQube, Container Security with Trivy and Grype, as well as DAST with OWASP-Zap and Arachni. The purpose of this study is to assess the efficacy of these instruments and derive valuable insights.

## Chapter 9: Conclusion

This thesis seeks to establish a connection between software development and cybersecurity. It provides valuable insights and actionable recommendations for organizations seeking to enhance their security measures within the dynamic landscape of modern software development.

## Software Development Lifecycle (SDLC)

The Software Development Life Cycle (SDLC) is a widely employed methodology within the software industry for the purpose of effectively executing and delivering software projects. The document outlines a comprehensive approach for the development, administration, replacement, and enhancement of a specific software application. The life cycle methodology provides a systematic approach to enhance software quality and streamline the development process.

The typical software development life cycle (SDLC) model adheres to a linear and sequential methodology, where each phase is carried out in a specified sequence prior to progressing to the subsequent phase. However, due to the advent of Agile and DevOps approaches, the Software Development Life Cycle (SDLC) has seen a significant transformation, becoming a more iterative and collaborative process.

### Agile Manifesto

In recent years, Agile methodology has begun to establish itself in the software development life cycle, in contrast to the waterfall methodology that most organizations and businesses had been accustomed to for many years.

Agile's beginnings date back to 1990, when a group of software developers began to question the efficacy of the Waterfall method. Although Waterfall is a structured and predictable method that simplifies project management and control, it can also be rigid and difficult to adapt to project changes.

A group of these developers created the Agile Manifesto, a set of guidelines, values, and ideas, in 2001 to help promote flexibility, collaboration, and rapid response to any changes.

The principles of the Agile Manifesto, according to the “Manifesto for Agile Software Development” [1] are the following:

- Satisfaction of the customer by delivering valuable software on time and on a regular basis is a high priority.
- Acceptance of changing requirements, even if they come late in the development process.
- Agile processes leverage change for the benefit of the customer's competitive advantage.
- Delivery of a working software, from a few weeks to a few months, with a preference for the shorter timescale.
- Throughout the project, businesspeople and developers must collaborate daily.
- Build projects around motivated people. Support and trust for a successful completion of a task.
- Face-to-face communication as is the most efficient and effective way of conveying information to and within a development team.
- The primary indicator of progress is functional software.

- Agile processes promote long-term development. Sponsors, developers, and users should be able to keep up the pace indefinitely.
- Continuous focus on technical excellence and good design improves agility.
- Simplicity, or the art of minimizing the amount of work done, is critical.
- Self-organizing teams produce the best architectures, requirements, and designs.
- The team reflects on how to become more effective at regular intervals, then tunes and adjusts its behaviour accordingly.

In essence, the Agile Manifesto establishes a structural foundation for the practice of Agile software development, prioritizing key principles such as cooperation, adaptability, and the fulfillment of client expectations. The widespread use of this tool throughout the software development community serves as evidence of its efficacy in optimizing development workflows and enhancing software excellence across several organizations.

## Continuous Integration and Continuous Delivery

Continuous Integration and Continuous Delivery, also known as CI/CD, is a best practice for DevOps and Agile. The CI/CD philosophy is a set of working principles, and a set of practices used by application development teams to release code updates more frequently and consistently.

Continuous delivery begins where continuous integration leaves off, automating application delivery to selected environments such as production, development, and testing. Continuous delivery is an automatic method of pushing code updates to these locations.

Continuous deployment can be implemented by a competent devops team with a strong CI/CD pipeline, in which application changes are run through the CI/CD workflow and passing builds are distributed straight to the production environment. Although continuous release isn't ideal for every business application, some teams choose to send it to production on a daily or even hourly basis.

The utilization of Continuous Integration and Continuous Delivery (CI/CD) yields a multitude of benefits for software development teams. Continuous Integration is a development approach that involves integrating code changes made by individual developers or teams into a centralized repository on a regular and automated basis. The integrated code changes are then immediately evaluated to ensure that they are appropriately merged and do not adversely affect the existing codebase. The primary objective of this practice is to detect integration issues early on, before they escalate into more significant challenges or cause delays in the development process. The use of CI/CD methodologies offers several benefits to development teams throughout the project's life cycle. For instance, developers have an opportunity to learn about new changes and their potential impact, enabling them to plan their work better and write code that is more reusable and easily testable. Furthermore, using automated tests makes it easier to identify errors and issues in the code, allowing for timely evaluation and resolution. This approach can also accelerate the development process, enabling teams to release new features and fix bugs more quickly, thereby enhancing customer

satisfaction and loyalty. CI fosters collaboration and communication among developers, as they often work together to merge code changes into a shared repository, resolving conflicts in a timely manner. Continuous Delivery reduces the risk of software failures and security breaches, as bugs and vulnerabilities can be detected and resolved quickly. Lastly, by reducing manual processes and increasing automation, organizations can save time and money on software development and delivery.

## DevOps

DevOps is a set of procedures and practices that integrates software development (Dev) and IT operations (Ops) in order to abbreviate the system development life cycle while providing features, patches, and upgrades frequently and consistently. It is a mindset of cooperation, communication, and automation among software engineers and IT workers.

According to Atlassian [2] because DevOps is an ongoing process, practitioners use the infinity loop to demonstrate how the stages of the DevOps lifecycle connect to one another. Even though it appears to run sequentially, the loop represents the need for continuous cooperation and iterative development throughout the entire lifetime.

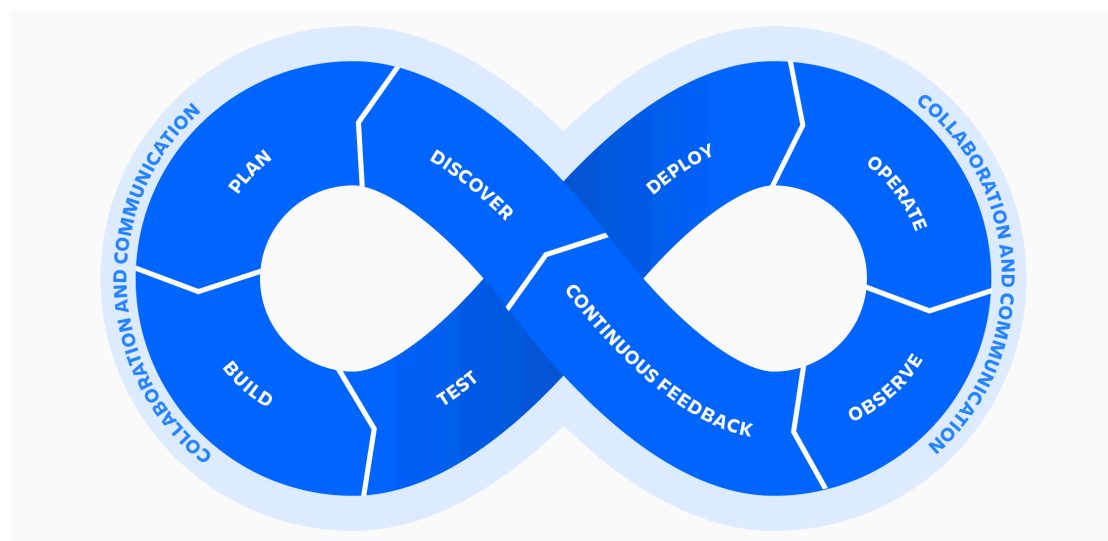


Figure 1: DevOps

There are many reasons that drive to the growing use of DevOps. First, rather than being created as a custom system or a shrink-wrapped product, software is increasingly being provided as a service over the Internet. This integrates processes into the product, driving service quality demands. [3]

As stated by Jabari and et al, DevOps is associated with rapid software development. DevOps broadens Agile's principles by offering a practical expansion to current agile activities. DevOps, for example, can achieve agile goals such as decreasing teamwork

delays and extending agile principles to the entire software supply chain by focusing on communication and collaboration between developers and administrators rather than tools and procedures. This point of view, which emphasizes the relationship between DevOps and agile methodologies, is not completely shared by the research community, as explained [4].

Overall, DevOps methodology provides a variety of benefits on the SDLC. DevOps provides improved collaboration between development and operation teams leading to more easy adaptation of changes, faster and more efficient deliveries. Also, the increased automation helps the team to build and deploy the project easier reducing the risk errors and improving overall quality, and have the customers satisfied too by finding faster resolution of issues. The greater visibility and control over the SDLC allow more effective monitoring and management of software projects. [3]

## Pipeline

A CI/CD pipeline is a set of automatic procedures that must be completed prior to releasing a new release of software. These pipelines use automation to improve software dissemination throughout the software creation process. [5]

CI/CD pipeline the creation, testing, production, and tracking phases of the software development lifecycle. This leads in faster and higher-quality code creation. Although it is possible to execute each stage of the CI/CD pipeline manually, the true advantages of these pipelines are realized through automation. [5]



Figure 2: CI/CD

The CI step includes regularly merging code changes into a common repository. This enables developers to test the code, detect and correct mistakes during the initial stages of development, before they increase bigger and take longer to correct. The code is built, tested, and validated in the CI stage using different tools and methods such as static analysis, unit tests, and integration tests.

The CD step includes the deployment on a stage environment or on the production of the code with all the changes that implemented under continuous integration.

Depending on the organization's rules and processes, the CD step involves deploying the code modifications to production or staging environments. The code is packaged, configured, and deployed using automatic scripts and tools during the CD step. This leads to the deployment procedure being constant, dependable, and repeatable.

CI/CD pipeline is the foundation of a DevOps approach, bringing together coders and IT operations teams to release software. As custom apps become more important in

how businesses distinguish themselves, the rate at which code can be published has become a competitive differentiator. [5]



# Application Security Pipeline for CI/CD

## Information Security

Information security is the practice of safeguarding data and information assets from illegal access, utilization, disclosure, modification, or loss. The subject matter at hand comprises both intangible and physical assets, spanning a wide range of concerns including network security, endpoint security, data protection, and user access restrictions.

Information security encompasses the protection of all assets related to information, whereas application security specifically focuses on safeguarding software applications and the associated data they contain. Application security is a subset of information security and a crucial element of a comprehensive information security initiative.

## Application Security

The methods and techniques used to protect software apps from possible threats and vulnerabilities are referred to as application security. This includes safeguarding against malicious players' illegal entry, tampering, and data stealing. Of course, the metrics that must be implemented arise by the risk associated with the application use. Controls and measurements can be applied to the application (its processes, components, software, and outcomes), its data (configuration data, user data, and organizational data), and all technology, processes, and players engaged in the application's life cycle [6].

As is defined in “ISO/IEC 27034 – Information Security – Security techniques – Application Security” [6]. standard, the scope of application security and the data/processes that should be protected are the following:

- Business context: The term "business context" encompasses the various guidelines, rules, and limitations related to the business field in which an organization operates.
- Regulatory context: The term "regulatory context" refers to the set of laws, rules, and established standards that begin in a particular region or authority and have an impact on the characteristics of the application or the way it processes data. Consider the risks associated with adhering to various national rules in nations where the same application is used.
- Application life cycle processes: The phases that software applications go through from inception to retirement are referred to as application life cycle procedures (Requirements, Design, Architecture, Coding, Testing, Deployment, Maintenance, Support, Retirement).
- Processes involved with the application: An application's processes relate to the numerous activities and duties required to develop, deploy, and manage the application.
- Technological context: The technological context is the collection of technological components and products that support an organization's essential data and specifications.

- Application Specifications: Hardware, security, application, client terminal and back-office specifications.
- Application data: Critical Application information.
- Organization and user data: Critical organization information.
- Roles and permissions: Crucial identification administration and permissions data.

Given the rising dependence on software applications to conduct important business tasks and the increasing complexity of cyber assaults in today's world, application security is an essential component of any organization's cybersecurity strategy.

## Security meets DevOps

DevSecOps is an approach that integrates security practices into the DevOps methodology, which places emphasis on collaboration among development, operations, and security teams to enhance the speed and efficiency of software development and deployment. The increasing intricacy and frequency of cyber-attacks and data breaches have underscored the importance of security in the field of software development, hence demanding the adoption of DevSecOps practices.

Historically, security has been seen as a distinct entity from development and operations, wherein security controls and evaluations are often conducted subsequent to the creation and dissemination of software. However, this particular approach may lead to the identification of security vulnerabilities at a later stage in the development process, so making their resolution more challenging and expensive.

The integration of security concerns into every phase of the software development lifecycle, including planning, coding, testing, and release, is made possible by the adoption of DevSecOps, which incorporates security into the DevOps process. This practice facilitates the identification and resolution of security vulnerabilities prior to their exploitation by unauthorized individuals.

According to the article "Challenges and Solutions When Adopting DevSecOps: A Systematic Review," [7] DevSecOps can assist organizations in improving their security posture, increasing the speed and quality of software development, and better aligning business objectives with IT operations.

Of course, there are many obstacles that an organization faces when adopting the DevSecOps methodology, such as resistance to change, a lack of understanding and comprehension of the methodology, and difficulty incorporating security into the software development process.

The article provides a comprehensive overview of the challenges that organizations face when implementing DevSecOps and offers various solutions to overcome these challenges, such as developing a security culture, encouraging team collaboration, and investing in automation and continuous monitoring tools. The writers also emphasize the significance of evaluating the efficacy of DevSecOps efforts and constantly refining the process to ensure that security is incorporated throughout the software development lifecycle.

## Challenges on adopting DevSecOps

As previously stated, the challenges associated with adopting DevSecOps include resistance to change, a lack of awareness and understanding of the methodology, problems in applying security into the software development process, and the need to manage complexity and scale as DevSecOps is implemented, according to the article "Challenges and solutions when adopting DevSecOps: A systematic review." [7]. Other obstacles include the shortage of skilled workers and the need to balance security standards with company objectives.

In more detail the obstacles are:

- Resistance to change: This challenge refers to some people or teams within an organization's hesitancy to embrace new methods, such as DevSecOps, due to a dread of the unknown, a lack of confidence, or a lack of top management buy-in. This obstacle can be surmounted by effectively conveying the advantages of DevSecOps and involving all parties in the adoption process.
- Lack of consciousness and comprehension of the methodology: This issue alludes to workers' dearth of information and comprehension of DevSecOps, especially those outside of IT and security departments. Organizations can surmount this issue by giving workers with training and education and fostering an atmosphere of constant learning and development.
- Difficulty incorporating security into the software development process: This issue alludes to the need to seamlessly and efficiently incorporate security into the software development process. This problem can be solved by adopting automation and constant monitoring tools, encouraging cooperation among development, operations, and security teams, and employing a "shift left" strategy to security in which security concerns are handled early in the development process.
- Managing complexity and scale: The need to handle the complexity and scale of DevSecOps deployments, especially in big and complicated companies, is referred to as this issue. This challenge can be met by beginning small and progressively growing up, employing agile methods, and ensuring that security is incorporated throughout the software development process.
- Skilled personnel: This issue alludes to the need for skilled personnel who are familiar with and knowledgeable about DevSecOps methods and tools. Organizations can meet this challenge by investing in training and education initiatives, employing experienced employees, and fostering an atmosphere of constant learning and growth.
- Balancing security requirements with business goals: This issue alludes to the need to balance security requirements with business goals, which is especially important in companies with conflicting objectives. This problem can be solved by engaging all stakeholders in the development process, ranking security needs based on risk, and constantly assessing and refining the DevSecOps process to guarantee alignment with business goals.

Overall, organizations can address these challenges by implementing DevSecOps in a complete manner, which includes tackling technical, cultural, and organizational problems as well as constantly evaluating and refining the process. [7]

# Vulnerabilities

## Definition

“A vulnerability is a hole or a weakness in the application, which can be a design flaw or an implementation bug, that allows an attacker to cause harm to the stakeholders of an application. Stakeholders include the application owner, application users, and other entities that rely on the application.” OWASP

A vulnerability within the realm of computer security pertains to a deficiency or imperfection in a system that may be exploited by an assailant to illicitly gain entry, engage in unlawful actions, or inflict harm upon the system. Vulnerabilities may manifest in several components of a system, including its software, hardware, network infrastructure, and human factors.

One of the primary approaches employed to identify vulnerabilities is through the practice of security testing, encompassing several activities such as penetration testing, vulnerability scanning, and code review. Once a vulnerability is identified, it may be classified according to its severity, impact, and exploitability.

Vulnerabilities can arise due to several sources, encompassing programming errors, design flaws, configuration inadequacies, and organizational vulnerabilities. Additionally, it is possible for someone to gain access to a system by exploiting third-party components or interfaces, therefore making the system vulnerable to potential flaws present in other interconnected systems.

The act of exploiting vulnerabilities can result in several consequences, including but not limited to the unauthorized acquisition of sensitive data, the compromising of system integrity or accessibility, and potential financial or societal damage. Therefore, it is imperative for organizations to acknowledge, prioritize, and rectify vulnerabilities promptly, employing strategies like as patching, configuration hardening, and user training. In addition, the implementation of security measures such as firewalls, intrusion detection systems, and access restrictions can effectively mitigate the exploitation of vulnerabilities.

## Application Vulnerabilities

An application vulnerability refers to a shortcoming or fault found in a software program that may be exploited by malicious actors to gain unauthorized access to sensitive information, disrupt regular operations, or carry out malicious activities on the system.

Vulnerabilities may arise due to deficiencies in program design, coding inaccuracies, configuration issues, or the use of obsolete software frameworks. Adversaries have the ability to exploit these vulnerabilities through the utilization of diverse techniques, including buffer overflow, SQL injection, cross-site scripting, and other forms of assaults.

The presence of issues in applications is a significant security concern for both businesses and individuals, since these vulnerabilities may be manipulated to illicitly obtain sensitive information, compromise the stability of systems, and inflict financial or societal damage. In order to promptly address vulnerabilities, it is imperative to regularly perform security testing and administer patches.

### *OWASP Top 10 Application Vulnerabilities 2022*

The Open Web Application Security Project's OWASP Top Ten is a list of the most important web application security threats. The list is revised on a regular basis to represent the changing threat environment and is widely used to rank security efforts by coders, security experts, and organizations. [8]

#### **1. Broken access control**

Unauthorized users may gain access to confidential data or perform activities that they should not be able to perform if access control is not properly implemented. For example, if a user gains unauthorized access to another user's account or information, this is considered a failed access control risk. This security risk occurs when a system or program fails to properly impose restrictions on what resources a person can access or the activities they are permitted to perform. Inadequate access control techniques, incorrect configuration, or faulty logic in the software program can all contribute to this.

##### *How to prevent this security issue?*

Access control is implemented in a secure server environment, where attackers cannot modify data. According to OWASP in order to prevent broken access control a lot of protection methods are proposed:

- API rate restriction and controller access
- Making access denial the default setting, unless the resource is public
- Enforcing business limits within the application
- Disabling directory listings for web servers
- Reusing access control mechanisms throughout the application

#### **2. Cryptographic failures**

Cryptographic failures refer to security weaknesses that arise from the improper implementation or use of cryptographic techniques.

##### *How to prevent this security issue?*

The prevention of cryptographic failure relies on the application's functionality and the nature of the data that is handled. Several aspects should be considered to protect data effectively. This includes classifying the data based on applicable laws, regulations, or business requirements, and storing only necessary data, which should be discarded once the task is completed. Additionally, it is essential to encrypt all data during transmission and at rest and avoid using outdated protocols to transfer sensitive data. These are just some of the measures for preventing cryptographic failure, with more detailed guidance available in OWASP reference guides.

### 3. Injection

Injection attacks are frequently used to exploit vulnerabilities in web apps, but they can also impact other kinds of software.

There are several types of injection attacks, including:

- SQL injection: An attacker injects malicious SQL code into a program, usually via an online form or user input area. This gives the attacker the ability to run random SQL queries and possibly obtain access to private data.
- In a command injection attack, an intruder injects malicious code into a program, which is then run as a system command. This gives the attacker the ability to run arbitrary instructions on the machine and possibly acquire control of it.
- Cross-site scripting (XSS): A cross-site scripting attack involves an attacker injecting malicious code into an online website, which is then performed in the user's computer. This allows the attacker to take user information or conduct activities on the user's behalf.

*How to prevent this security issue?*

In order to mitigate the risk of injection attacks, which have the potential to exploit many software applications, including online applications, it is imperative to incorporate effective security protocols. The recommended security measures encompass several key aspects, such as rigorous input validation, utilization of parameterized statements for database interactions, adherence to the principle of least privilege to restrict access, deployment of a web application firewall (WAF), regular application of security patches, implementation of robust session management techniques, utilization of Content Security Policy (CSP) headers, provision of developer education in secure coding practices, continuous monitoring for any signs of suspicious activities, and the establishment of a fail-safe mode or intrusion detection system. The collaborative endeavors mentioned will serve to protect against injection attacks and enhance the security of software applications.

### 4. Insecure Design

Insecure application design refers to the various flaws in the design of software apps that make them vulnerable to various kinds of cyberattacks. Attackers can leverage vulnerabilities in an application to obtain unauthorized access to private data, alter or delete data, or disrupt the application's operation.

*How to prevent this security issue?*

To promote a culture of security, it's important to use a secure design methodology that evaluates potential threats and ensures that code is designed and tested against known attack methods. This can help prevent security breaches and minimize the risk of vulnerabilities being introduced into the system.

One way to achieve this is by partnering with application security professionals to help evaluate and design controls around privacy and security, and by using a secure development lifecycle to ensure that security measures are incorporated throughout the software development process.

Additionally, it's recommended to use components and design patterns that are known to be secure and to apply threat modeling to critical areas such as access controls, key data flows, business logic, and authentication. By doing so, organizations can better identify and address potential security threats before they can be exploited.

#### 5. Security Misconfiguration

When security settings or configurations are not applied correctly or are left in their default state, security misconfiguration happens. As a result, the application or system is susceptible to assault, as attackers can take advantage of these misconfigurations to obtain unauthorized access, pilfer confidential data, or carry out other malicious activities.

*How to prevent this security issue?*

According to OWASP, to prevent security breaches, it's important to start with a thorough security configuration process that can be repeated consistently across systems, ideally through automation. This can help ensure that new environments are secured appropriately with every deployment.

One way to achieve this is by implementing a security hardening process that can be repeated and automated to minimize errors and ensure consistency. Additionally, it's recommended to use only the features and components that are necessary for the system's operation, as unused or unneeded components can increase the attack surface and pose security risks.

Finally, deploying an automated process to review security settings across different environments can help identify any misconfigurations or vulnerabilities that may have been introduced. This can help ensure that security measures are implemented consistently and effectively across the organization's infrastructure.

#### 6. Vulnerable and Outdated components

Vulnerable and obsolete components are software components, libraries, or frameworks used in apps that have known security flaws or are no longer maintained by the manufacturer, leaving them vulnerable to cyber-attacks.

*How to prevent this security issue?*

Establishing a process for managing patches can help reduce the risk of security breaches by addressing vulnerabilities before they can be exploited. This process should involve removing any unused or unnecessary libraries,

components, frameworks, documentation, and files, which can help reduce the attack surface of the system.

Additionally, it's important to continually monitor and maintain an inventory of both server-side and client-side components to ensure that they are up to date and secure. It's also recommended to use only official libraries and sources through secure links to minimize the risk of downloading malicious or tampered-with code.

Finally, organizations should monitor for any unsupported libraries or components that are no longer maintained or have reached their end-of-life, as these can pose significant security risks if they contain vulnerabilities that are not being addressed.

#### 7. Identification and authentication failures

Identification and authentication failures refer to instances where an individual or system is not correctly identified or authenticated, leading to unauthorized access or activities. Identification is the process of providing a username or other identifier to gain access to a system, while authentication is the process of verifying that the user is who they claim to be, typically through a password or other authentication mechanism.

*How to prevent this security issue?*

According to OWASP, To prevent security breaches, it's important to ensure secure storage and retrieval of passwords. One way to achieve this is by implementing multi-factor authentication, which adds an additional layer of security beyond just a password.

Furthermore, it's crucial to avoid using default credentials, especially for administrative accounts, as these are often known to attackers and can easily be exploited. It's also recommended to limit exposure to account enumeration, which involves using trial and error to guess valid usernames and passwords by limiting the number of login attempts or locking out users after a certain number of failed attempts.

By implementing these measures, organizations can help protect against unauthorized access and reduce the likelihood of security breaches.

#### 8. Software and data integrity failures

Software and data integrity failures refer to instances where the accuracy, completeness, or consistency of software code or data is compromised. These types of failures can occur due to various factors, including human error, hardware or software malfunctions, cyber-attacks, or other external events.

*How to prevent this security issue?*

To prevent security breaches, it's essential to verify the authenticity and integrity of software updates. This can be achieved by using digital signatures



or other verification methods that ensure the updates originate from expected sources and are delivered without any alterations.

In addition to verifying software updates, it's important to ensure that any third-party libraries or dependencies used in the software also come from legitimate sources. This can help prevent the introduction of malicious code into the system.

Another crucial step in prevention is to verify that third-party resources are free from vulnerabilities. This can be done using automated security tools specifically designed for the software supply chain, which can help identify and mitigate any potential risks or vulnerabilities.

#### 9. Security logging and monitoring failures

Security logging and monitoring failures refer to instances where an organization fails to properly implement and maintain security logging and monitoring processes, which can leave their information systems vulnerable to security breaches.

*How to prevent this security issue?*

To prevent security breaches, the primary focus should be on setting up security logging and monitoring capabilities across applications. Developers should implement appropriate security controls, such as login, access control, and server-side validation checks, and ensure that any failures are logged with user context to identify and analyze any malicious or suspicious activities.

Moreover, it's important to generate logs in a format that can be easily read by log management tools. Enabling monitoring and alerting mechanisms for identifying suspicious activities is also recommended. Finally, having a well-defined incident response and mitigation plan in place can help organizations respond swiftly and effectively in the event of a security incident.

#### 10. Server-side request forgery

In a Server-Side Request Forgery (SSRF) attack, the attacker takes advantage of the server's functionality to gain unauthorized access to resources or alter them. The attacker's focus is on exploiting an application that can import data from URLs or read data through URLs. They may manipulate the URLs by either replacing them with different ones or tampering with URL path traversal to achieve their goals.

*How to prevent this security issue?*

According to OWASP, SSRF can take place at both the network and application levels. To safeguard networks, it's advisable to use network segmentation to isolate remote resources and prevent nonessential traffic with "deny-by-default" policies.

Regarding application protection, some recommended methods include thoroughly sanitizing, validating, and filtering data inputs. Additionally, disabling HTTP redirection at the server level and verifying that server responses match expected results are crucial. It's essential to avoid transmitting raw server responses to clients.

## Docker Image Vulnerabilities

Docker is a containerization system that facilitates the organization and execution of software applications in a portable and resource-efficient manner. This technology enables software developers to create and implement programs in a uniform and reliable manner across several computer environments, encompassing development, testing, and production stages.

Docker containers consist of a runtime environment, an image, and the executing processes contained within them. The runtime is responsible for managing the resources of the container, such as the CPU, memory, and storage. The fundamental representation encompasses the application code and its associated dependencies. Processes refer to programs that function within a container and engage with the image and runtime components.

According to the study of Martin et al [9] there are five docker image vulnerability categories:

1. **Improper configuration.**  
Docker's default setup isolates containers and limits their access to the host, making it reasonably safe. Certain settings sent to the Docker daemon or the command starting a container, on the other hand, might grant the host expanded access and render it vulnerable to assaults. These options can cause the isolation property to be broken, hence they should only be used with trusted containers. The Center for Internet Security has developed a Docker Benchmark that lists the settings that should and should not be used when running containers as isolated apps using Docker. A Docker host hosting containers may be made more secure by following these guidelines and best practices.
2. **Vulnerabilities in the image distribution, verification, decompression, storage process.**  
Docker Hub's design is comparable to that of a package repository, leaving it subject to the same vulnerabilities as package managers. These flaws involve the processing, storage, and uncompression of potentially malicious code by the Docker daemon with root access. Package management attacks are conceivable if an attacker has control of a portion of the network between the Docker host and the repository. A successful attack would allow her to make her image downloaded on docker hosts, leading to compromised images that can exploit vulnerabilities in the extraction process.

The article also highlights Docker-specific vulnerabilities, such as those in the extraction process and connected to the automated build chain. The Docker

hub's automated builds and webhooks are critical components of this distribution strategy. They lead to a pipeline in which each component has complete access to the code that will be used in production. Account hijacking, interference with network traffic, and insider assaults are examples of compromise tactics in this architecture. The configuration adds multiple external intermediary stages to the code route, each with its own authentication and attack surface, increasing the overall attack surface.

3. Vulnerabilities within the images themselves  
This increases the attack surface of the Docker images, especially since the DevOps movement allows developers to package their own applications, potentially mixing development and production environments and leaving vulnerabilities. Additionally, outdated versions of packages are often present in the provided images, which increases the risk of exploitation. Attacks can come from both outside and inside the container, and if the container has an entry point, exploitation of vulnerabilities is possible. The Docker Security Scanning feature can help mitigate some of these risks by scanning each layer of the image and identifying known security vulnerabilities, but it has limitations such as being available only for private repositories and the cost of the service.
4. Vulnerabilities related to Docker or libcontainer  
The paper discusses many file-system isolation security flaws in Docker and libcontainer, including chroot escapes, path traversals, access to special file systems, container escalation, and privilege escalation. These vulnerabilities have been fixed in various Docker versions; however, the report warns that even if a container process is granted root capabilities, it may still be able to access the full host filesystem, which might result in delayed arbitrary code execution with root privileges. The article also discusses the use of Mandatory Access Control (MAC) to enforce constraints on container processes but notes that the default Apparmor policy for Docker containers, known as "docker-default Apparmor policy," could be improved because it primarily functions as a whitelist, granting containers full access to network devices and file systems, with only a limited number of deny directives acting as a blacklist.
5. Vulnerabilities related of the Linux kernel  
Because containers operate on the same kernel as the host, they are vulnerable to kernel attacks. An attacker can utilize a kernel vulnerability within a container to break out of the container and infiltrate the host, resulting in an isolation breach, integrity breach, and data disclosure.

### Pipeline Vulnerabilities

A pipeline vulnerability is a fault or weakness in a system's data or communication pipeline that attackers can exploit to gain unauthorized access to sensitive data or execute destructive activities.

Pipeline vulnerabilities can be caused by several factors, including unsafe coding techniques, unpatched software or operating systems, insufficient authentication procedures, or improper network or system setup. By introducing malicious code or instructions into the pipeline, intercepting or modifying data in transit, or obtaining privileged access to system resources, attackers can exploit pipeline vulnerabilities.

Organizations can install safety measures like as access restrictions, encryption, firewalls, intrusion detection and prevention systems, and frequent security audits and testing to reduce pipeline vulnerabilities. It is also critical to keep up with the newest security patches and upgrades for any software and systems in use, as well as to teach personnel on secure coding standards and data protection best practices.

## Vulnerability detection approaches throughout the SDLC

### Testing Types

There are several detection vulnerability approaches that can be used throughout the Software Development Life Cycle (SDLC) to ensure the security of software applications.

The most common approaches are:

*Threat Modeling:* Threat modeling is an organized method that identifies possible security threats and vulnerabilities, quantifies their severity, and prioritizes mitigation measures to protect IT resources. It is a must-do for any organization that wishes to protect its systems, apps, networks, and business processes against a wide range of threat vectors. According to fortinet.com [10], the threat modeling process involves a sequence of interdependent steps such as describing the issue, developing a list of assumptions, and verifying the techniques for dealing with the risks.

Threat modeling creates a clear "line of sight" throughout a project, justifying security efforts and allowing for informed decision-making regarding application security threats. According to OWASP, the threat modeling method gathers, organizes, and analyzes all information that influences an application's security, resulting in a prioritized list of security enhancements to an application's idea, requirements, design, or implementation. Furthermore, the threat model enables rational security decisions to be made with all available information.

*Static Code Analysis:* Static code analysis is the process of examining an application's source code without running it. This method is typically used throughout the SDLC development process and can assist in identifying possible security flaws in the code.

*Dynamic Code Analysis:* The examination of dynamic code occurs while the program is executing. This method is typically used during the SDLC testing phase and can assist in identifying possible security vulnerabilities in the application's runtime environment.

*Penetration Testing:* Manual penetration testing includes imitating an attacker's behavior and is classified as black box testing since it does not require knowledge of implementation specifics. This phrase refers to the analysis of the system without knowledge of its internal state and structure. The goal of these tests is to uncover any type of vulnerability, from minor implementation issues to major design faults. [11]

*Fuzz Testing:* Fuzz testing is flooding the application with enormous volumes of random input data in order to examine its robustness to unexpected inputs. This method is typically used throughout the SDLC testing phase and can assist in identifying potential security issues linked to input validation and handling.

*Software Composition Analysis:* Software Composition Analysis (SCA) is a software engineering practice that involves analyzing custom-built software applications to identify any open-source software embedded within them. The purpose of this analysis is to evaluate any potential vulnerabilities and ensure that the software

complies with licensing requirements. The practice has gained significance with the growing use of open-source software, which has allowed developers to rapidly add functionality to their proprietary software. SCA is a subset of the application security testing (AST) tool market, specifically focused on managing the use of open-source components.

This thesis will go through static, dynamic, and software composition analysis while skipping over thread modelling, fuzzy testing, and penetration testing.

## Static Code Analysis

According to OWASP [12], Static Code Analysis, also called as Source Code Analysis, is often conducted at the Implementation phase of a Security Development Lifecycle (SDL) as part of a Code Review or white-box testing. Static Code Analysis tools are used to examine 'static' (non-running) source code to discover possible vulnerabilities. These technologies use a variety of approaches, such as Taint Analysis and Data Flow Analysis, to identify potential security concerns.

ISO 27034 [13] suggests that project teams do static code analysis on their source code as a scalable method of assessing security code and assuring compliance with secure coding guidelines established by the security team lead and adviser. It should be highlighted, however, that static code analysis alone may not be sufficient to complete a thorough security examination. As a result, the security team and advisers should be aware of the capabilities and limits of static analysis tools and, if necessary, augment code review activities with other tools or human review. A lightweight static analysis is performed as part of the SDL during code check-in using the IDE.

Static analysis tools may be used in CI/CD to find vulnerabilities and verify that secure coding practices and rules are adhered to. Furthermore, these tools may be incorporated into a developer's work environment and used to create replacement proposals for vulnerable code with more secure code (for example, deprecated library functions). [11].

In their 2019 article [14], Nora Tomas, Jingyeue Li, and Huang Huang emphasized the significance of delivering working software frequently as a fundamental principle of agile software development. To achieve this, developers need to be able to test and verify their code quickly and effectively. Automated testing and code quality tools, which may include static analysis tools, can provide immediate feedback on potential errors or defects in the code, thus facilitating the attainment of this goal. Static analysis tools can also be integrated into the build and deployment process to automatically detect potential issues and ensure high-quality code. While the article does not focus specifically on static analysis, it acknowledges the potential advantages of using such tools in an agile environment.

On the other hand, Thorsten Ragnau, Remco Buijtenen, Frank Fransen and Fatih Turkmen in their article on 2020 [15], suggests that while static code analysis is an important practice in DevSecOps, it is not sufficient to detect all security vulnerabilities in a system. Static analysis can only identify vulnerabilities that can be directly derived from source code, which is a small subset of the most common

vulnerabilities in web applications. The article notes that dynamic security testing, which involves attacking a system in a manner like actual hackers, can identify a much broader range of vulnerabilities. While there is literature available on how to execute dynamic tests consistently and reproducibly, there is less information on how to integrate dynamic testing into the CI/CD pipelines commonly used in DevOps.

## Dynamic Code Analysis

Dynamic Code Analysis, also known as Dynamic Application Security evaluating (DAST), is a way of evaluating an application while it is operating to uncover potential security flaws, according to OWASP [16]. To "fuzz" the program, DAST tools employ a database of known security vulnerabilities and malicious inputs, such as input strings with odd lengths, negative and huge positive values, and unexpected input data. The DAST tool notes the detected vulnerability if the program responds negatively to a given input. DAST is often used during the testing phase of the Software Development Lifecycle (SDLC), after the application's code has been successfully built and deployed to a test or staging environment. DAST scans can run numerous times per day when iterative builds occur using continuous integration/continuous delivery (CI/CD) workflows. DAST happens following penetration testing that do not involve internal security knowledge in many business software security initiatives [17].

The study "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines" [15] states that dynamic code analysis is important in the continuous security testing process. Dynamic code analysis detects security flaws that may occur from the interplay of multiple components or external dependencies by monitoring code execution within the CI/CD pipeline. The article explores integrating dynamic security testing tools into the CI/CD pipeline to automate the identification of security risks throughout the software development lifecycle. This connection provides continuous evaluation of the application's security posture and fast remedy of vulnerabilities.

The authors emphasize the advantages of employing dynamic code analysis approaches, such as runtime vulnerability discovery, real-time security feedback, and increased visibility into program behaviour in a variety of contexts. Overall, the article demonstrates how including dynamic security testing tools into CI/CD pipelines may improve software application security.

On the other hand, according to the article "Challenges and Solutions When Adopting DevSecOps: A Systematic Review," [7], in the context of DevSecOps (Development, Security, and Operations), dynamic analytic tools have several drawbacks. Here's a rundown of the points raised:

- Running software for dynamic analysis: To do the analysis, dynamic analysis tools require the program or code to be executed. This entails creating, installing, and configuring the software, which can be difficult in a DevSecOps setting with regular code releases.
- Manual work and setup: Dynamic analytic tools can need extensive manual effort to set up and run efficiently. Manual setup adds complexity and can be time consuming.

- Time required for analysis: Dynamic analysis techniques, such as Static Application Security Testing (SAST) tools, often take a longer time to perform. The analysis process can be time-consuming, which might impact the speed and frequency of releases in a DevOps environment.
- Testing scenario constraints: Dynamic analysis tools may have limitations in terms of the spectrum of testing scenarios they can successfully cover. The number of vulnerabilities and testing scenarios that may be detected is determined on the type of dynamic analysis tool employed.

These constraints make the incorporation and acceptance of dynamic analytic tools in DevSecOps techniques difficult. In a DevOps context, the aforementioned challenges may have an influence on the speed and frequency of software releases. Despite these limitations, dynamic analysis methods can nevertheless detect security flaws, emphasizing their value in the overall security evaluation of software applications. In the final chapter of this thesis, we will look at the benefits and drawbacks of DAST tools.

### Software Composition Analysis

The technique of integrating smaller, independent things into a bigger, more complicated object or system is known as composition in software engineering. This is accomplished by defining classes with instance variables that refer to one or more objects from other classes. Composition is used in both functional and object-oriented programming. Composition is exemplified in functional programming with functions that accept one input, "foo," and return another input, "bar," and another function that takes "bar" as input and returns "blah." There is also a combined function that accepts "foo" as input and returns "blah". Composition has the virtue of allowing code to be reused without creating a "is-a" connection, which enhances encapsulation and makes the code easier to maintain.

Composition is also utilized often in software components to generate clear, user-friendly APIs. When a class is created, the classes it references can either become part of the API or be hidden. If a class does not have any access modifiers, it becomes package-private, which means that it may only be accessed within its own package and is not part of the API. External clients can only communicate with the software component through a public class that leverages the package-private class in composition.

The fundamental advantage of composition is that it allows for code reuse without the need for an is-a connection, like inheritance does. This method improves encapsulation and makes code maintenance easier. To generate clean and user-friendly APIs, composition is widely used in well-designed software components. When creating a class, you have the option of exposing the referenced classes as part of the API or keeping them concealed. If a class does not have any access modifiers, it becomes package-private, which means it may only be accessed within its own package and is not part of the API. External clients of your software component are ignorant of this class and can only communicate with it through a public class that uses composition to integrate the package-private class. [18]



The process of identifying and managing open source and third-party components used in software systems is referred to as software composition analysis (SCA). It entails examining the software's dependencies for known security flaws, licensing difficulties, or obsolete versions. SCA tools help to automate this process and deliver information to developers and organizations for them to manage possible hazards linked with the components utilized in their program.

The combination of Software Composition Analysis and DevSecOps places security at the center of software development. Organizations may proactively identify and mitigate possible risks associated with the usage of third-party components by introducing SCA into the DevSecOps pipeline, ensuring that their software applications are safe and comply with licensing requirements. This integration fosters a security culture and helps developers to create more secure and dependable applications.

### Container Security Analysis

Container security is a critical component of DevSecOps, necessitating constant security monitoring across development, test, and production environments. A new approach to the basic visibility, detection, and investigation procedures is required to protect containerized applications in a DevSecOps architecture [19]. Here are some methods to consider for DevSecOps container security analysis:

- Encourage developers, operations, and security teams to work together to ensure security is integrated in the software development process, with the objective of making it visible and automated [20].
- Improve productivity and security across the software development life cycle by introducing established procedures, extensive documentation, and agreed-upon automated processes [20].
- Integrate the multiple platforms, technologies, and procedures needed for application development, deployment, and operations to create a coherent and cohesive system known as DevSecOps [20].
- Utilize Kubernetes information like as labels and annotations to discover and categorize security insights automatically, allowing automated security detection inside the Kubernetes environment [19].
- For DevSecOps projects, use a common analytics platform that gives integrated insights across the whole tool chain and technology stacks, enabling for thorough analysis and monitoring [19].
- Improve the discovery, categorization, and resolution of security issues in apps, configurations, and container images by implementing effective vulnerability and configuration management strategies [19].

Vulnerability scanners for container images can be used to build secure container-based CI/CD operations as mentioned in the last chapter of this thesis.

### Continuous Vulnerability Management in DevSecOps

## Continuous Vulnerability Evaluation

In DevSecOps, vulnerability evaluation plays a crucial role in ensuring the security of software and systems throughout the development lifecycle.

According to *Challenges and solutions wen adopting DevSecOps* [7], several challenges can arise in the process of a continuous vulnerability evaluation. First, it is not commonly used, owing to practitioners' failure to do periodic vulnerability checks and team members' lack of information about continuous vulnerability assessment. Furthermore, the lack of a consistent technique for incorporating security measures into a DevOps pipeline blocks progress.

Another obstacle that hinders the integration of security and DevOps processes is the lack of compatibility between the two. This is mostly since security testing sometimes requires substantial human involvement and can be a time-consuming process, which contradicts the requirement for rapid software releases. The increasing intricacy, vulnerabilities, and dependence on external elements pose significant difficulties in ensuring security assurance. Consequently, software engineers often encounter challenges in balancing the speed of software releases with ensuring complete security. Similarly, corporations perceive the ongoing adoption of DevOps practices and the achievement of comprehensive security assurance as conflicting strategies.

In addition, the fast-paced nature of continuous deployments makes it difficult to thoroughly verify security requirements before software is shipped to the production environment. As a result, practical assessments of security requirements often go unperformed. This situation may be attributed to a lack of suitable tools and methods to carry out the assessment process effectively. The absence of appropriate resources contributes to the difficulty in conducting rapid security requirement assessments in a DevSecOps setting.

Organizations can implement continuous vulnerability evaluation inside a DevSecOps environment by following the suggested best practices mentioned:

- **Developer-centric approach:** Make sure that security tools and solutions emphasize the needs of developers by being user-friendly, simply understandable, and seamlessly integrated into their existing processes. This technique encourages developers to actively participate in the vulnerability evaluation process.
- **Prioritization of vulnerabilities:** Use a tool that can analyze vulnerabilities depending on their level of risk, resulting in more accurate results and fewer false positives. This technique encourages greater adoption and enables for concentrated efforts to fix significant vulnerabilities.
- **Use automation to speed up vulnerability detection and repair methods.** Organizations may incorporate security technologies smoothly into their development and CI/CD pipelines by introducing automation, hence speeding the vulnerability evaluation process.
- **Collaboration and knowledge sharing:** Foster collaboration among developers, operations, and security teams. Breaking down silos encourages effective communication, facilitates sharing of knowledge and insights, and enables joint efforts to identify and resolve vulnerabilities throughout the software development lifecycle.

- Continuous monitoring: Stay vigilant by keeping up to date with the latest vulnerabilities, tools, and fixes. Constantly monitoring for emerging security threats and seamlessly integrating appropriate measures into processes ensures an up-to-date security posture.
- Adoption of modern DevSecOps tools: Consider implementing interactive application security testing (IAST) as an alternative to traditional vulnerability scanning tools such as static application security testing (SAST) and dynamic application security testing (DAST). IAST provides real-time vulnerability management, goes beyond code analysis, and offers a broader perspective for comprehensive evaluation.

By embracing these best practices, organizations can improve the effectiveness and efficiency of continuous vulnerability evaluation in their DevSecOps initiatives. By prioritizing developers, utilizing automation, fostering collaboration, maintaining vigilance, and adopting modern tools, organizations can enhance their overall security posture and effectively mitigate potential risks.

### Continuous Vulnerability Treatment

Continuous Vulnerability Treatment in DevSecOps is a crucial aspect but it comes with its own set of challenges.

Tool-related challenges: Automation is a crucial necessity in DevSecOps, and businesses confront obstacles in acquiring and integrating several solutions that can enable continuous and consistent security measures. To avoid and mitigate security vulnerabilities throughout the development lifecycle, it is critical to ensure interoperability between security measures and the tools and techniques used in DevOps operations. [21]

Shift-left security: Shift-left security is the practice of incorporating security controls at every level of the application and infrastructure life cycles. This allows for the early detection and remediation of vulnerabilities, limiting the possibility of attackers exploiting them in production systems. Shift-left security demands a mentality and procedure shift, which may be difficult for enterprises to implement. [22]

Continuous security assessment: The process of incorporating vulnerability detection and patching into the release cycle. This approach aids in the rapid identification and remediation of common vulnerabilities and exposures (CVE), narrowing the window of opportunity for threat actors to exploit flaws in production systems. Implementing and maintaining appropriate continuous security assessment methods, on the other hand, might be difficult. [23]

People-related factors: While less studied, people-related factors play a crucial role in the successful adoption of DevSecOps. Organizations face challenges in developing a collaborative culture that fosters the fusion of development, security, and operations teams. Additionally, continuously developing new concepts and tools to support DevSecOps practices requires ongoing learning and skill development within the workforce. [7]

Organizations can overcome security challenges by adopting specific strategies. Firstly, they should automate security controls within the CI/CD pipeline to ensure consistent application of security measures throughout the software development lifecycle. This involves integrating compliance checks, vulnerability scans, and other security controls based on predefined policies triggered by various pipeline events. Secondly, organizations should take a product-agnostic approach by implementing a holistic security framework. This framework should cover critical dimensions of DevSecOps, including culture, continuous security traceability, real-time monitoring of the pipeline's security posture, and meaningful metrics for monitoring. Additionally, organizations can benefit from using a DevSecOps cyber range platform to practice security investigations and automate continuous security. Lastly, access to up-to-date threat intelligence information is vital, enabling organizations to stay informed about the latest tactics, techniques, and procedures used by attackers to exploit applications and infrastructure components. [22]

### Continuous Vulnerability Reporting

Vulnerability reporting in a DevSecOps continuous pipeline presents several challenges. Firstly, the fast pace of software updates and deployments makes it difficult to identify vulnerabilities in a timely manner. Continuous monitoring and scanning of software components throughout the pipeline are essential to address this challenge.

Secondly, automated scanning tools used to identify security flaws can produce inaccurate results, leading to false positives and false negatives. To minimize inaccuracies, organizations must invest in fine-tuning and validating these tools.

Integrating vulnerability reporting into an existing DevSecOps pipeline can be complex, requiring seamless integration with various tools and processes such as code repositories, build systems, deployment pipelines, and issue tracking systems. Ensuring smooth data flow and compatibility between different tools and processes can be a challenge.

As the scope of the pipeline expands in relation to larger projects or heightened development velocity, the quantity of vulnerabilities and security findings might become excessively burdensome. Effectively managing a substantial influx of vulnerability reports necessitates the implementation of strong procedures and automation to efficiently handle the heightened workload.

Prioritization and remediation of vulnerabilities is another challenge. With numerous reports, accurately prioritizing vulnerabilities becomes crucial. Organizations must establish a clear risk-based prioritization framework to address the most critical vulnerabilities first.

Effective collaboration and communication among developers, security teams, and operations teams are essential for addressing vulnerabilities efficiently. However, achieving smooth collaboration can be challenging due to different perspectives and priorities. Establishing effective communication channels and fostering a culture of security collaboration is necessary.

In conclusion, it is imperative that vulnerability reporting be not treated as a singular occurrence, but rather as an ongoing process, with continual monitoring being of utmost importance. The proactive identification and remediation of vulnerabilities may be facilitated by the implementation of frequent vulnerability scans and the integration of vulnerability management into the DevSecOps pipeline.

# Continuous Vulnerability Finding in DevSecOps – Case Study: OWASP WebGoat Application / Tools Compare

## Approach and all Set-up

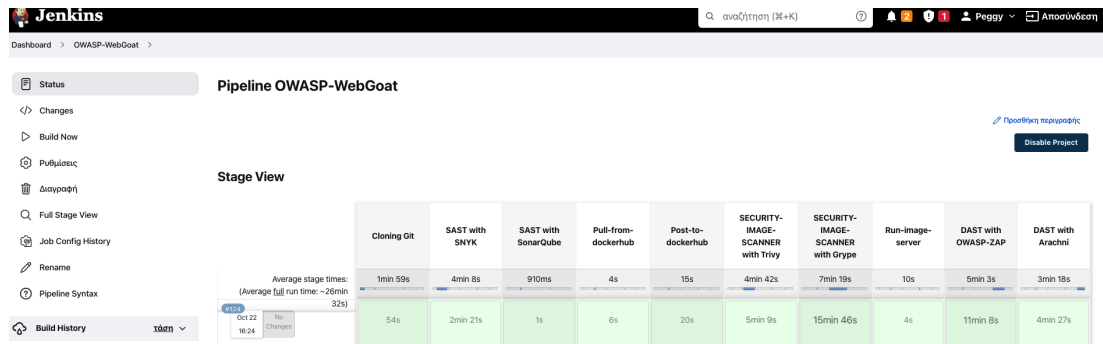


Figure 3: DevSecOps Jenkins pipeline

### Approach

As previously mentioned, tools play a critical role in the continuous vulnerability management process in DevSecOps. They help identify, assess and remediate vulnerabilities in an automated and efficient manner. However, there can be challenges associated with these tools.

One significant challenge is the integration of various security tools into the DevSecOps pipeline. Integrating tools with different interfaces, formats, or compatibility issues can impede the seamless incorporation of these tools into the existing CI/CD workflow. This can lead to delays, manual efforts, and potential gaps in vulnerability detection and treatment.

Another challenge lies in the accuracy of vulnerability scanning tools, which may produce false positives or false negatives. False positives can result in wasted time and resources spent on investigating non-existent vulnerabilities, while false negatives may leave security risks undetected. Striking the right balance between sensitivity and specificity in the tools is crucial to minimize false results.

As the DevSecOps pipeline scales and handles larger codebases and deployments, the performance and scalability of vulnerability treatment tools become critical. These tools need to handle increased volume and complexity without causing bottlenecks or compromising the speed of software delivery.

Furthermore, selecting and maintaining suitable tools for vulnerability treatment can be a challenge. With numerous options available, organizations must carefully evaluate the strengths, weaknesses, and costs of different tools. Regular monitoring, updates, and customization of tools are essential to ensure their effectiveness in detecting and treating vulnerabilities over time.

In this thesis, we are going to use an existing project named “OWASP WebGoat Application” by OWASP, in order to compare multiple tools for a continuous vulnerability management in a DevSecOps environment. A pipeline has been set up to coordinate the flow of the application.

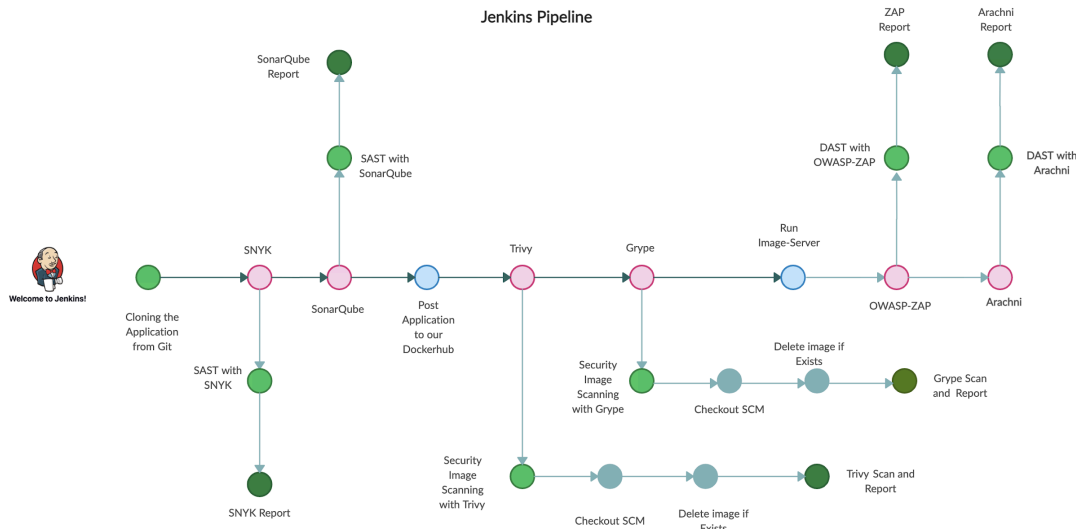


Figure 4: DevSecOps Jenkins approach

As can be seen from the image, a basic pipeline has been created using Jenkins, which in turn invokes other pipelines for security testing of the application. Jenkins has been installed on a Linux machine, which we will refer to as the Jenkins machine.

In the second step, an external pipeline is invoked, which utilizes the SNYK tool. SNYK performs static code analysis for security checks and generates a report. The third step calls an external pipeline that uses the SonarQube tool. SonarQube performs static code analysis for code quality checks, including security aspects, and generates a report. Both static analysis tools are installed in the same environment, on the application-server-1.

In the fourth step, the application is built into a Docker virtual environment and uploaded to DockerHub.

In the fifth and sixth steps, two external pipelines are invoked, which use the Gripe and Trivy tools respectively. These tools perform vulnerability scanning on the Docker image and generate reports with the detected vulnerabilities.

The next step involves deploying the application to the "application-server-1" using the Docker image that was previously created.

In the last two steps, external pipelines are called, which utilize the OWASP ZAP and Arachni tools respectively. These tools perform dynamic security analysis on the already installed application on the "application-server-1" and generate reports with any identified vulnerabilities. The purpose of this pipeline is to present a

comprehensive DevSecOps pipeline for an application but primarily to execute and compare tools of the same type.

All the tools will be further presented in this specific chapter and compared at the end based on certain criteria.

### Set Up

To create the pipeline, three virtual machines (VMs) were installed from scratch. These VMs use the Linux operating system with Ubuntu 18.04 version. They were installed in the VirtualBox environment, which was set up on Unix software. In more detail:

VM1 - Jenkins Machine:

This VM was set up for the installation and use of the Jenkins server. An image of Jenkins was installed on VM1, and it was deployed.

VM2 - Application-server-1:

This VM was set up for the installation and use of SNYK, SonarQube, ZAP, and Arachni tools.

Additionally, a Jenkins's agent <sup>1</sup> named "ubuntu" was set up on this VM to serve parallel pipelines.

VM3 - Application-server-2:

This VM was set up for the installation and use of Grype and Trivy tools. Additionally, a Jenkins's agent named "ubuntu2" was set up on this VM to serve parallel pipelines.

## Jenkins

Jenkins is an open-source automation platform that enables project continuous integration and delivery. It is a robust program that can handle any type of build or continuous integration and is compatible with a variety of testing and deployment systems. Organizations may use Jenkins to automate the software development process and include development life-cycle operations such as build, document, test, package, stage, deploy static analysis, and much more [5].

Jenkins's widespread adoption, with over 147,000 active installations and over 1 million users worldwide, and its interconnectivity with over 1,000 plugins that allow it to integrate with most development, testing, and deployment tools [6], are two of the features that set it apart from other Continuous Integration tools.

Using pipelines, Jenkins provides a simple approach to build up a continuous integration or continuous delivery environment for practically any combination of languages and source code repositories, as well as automate other normal

---

<sup>1</sup> In the context of Jenkins, agents, also known as Jenkins slaves, are worker nodes that perform tasks as part of a Jenkins build or automation process. These agents are separate machines or computing environments that can be connected to a Jenkins master, allowing distributed execution of jobs and providing scalability for Jenkins-based Continuous Integration (CI) and Continuous Deployment (CD) pipelines.



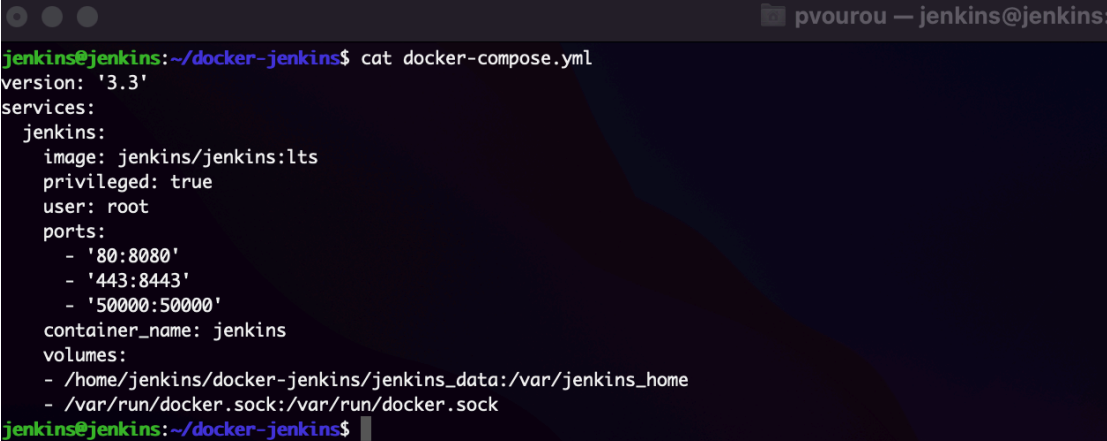
development chores. Jenkins can also easily distribute work over many computers, allowing for speedier builds, testing, and deployments across various platforms [7].

Jenkins uses pipelines to create a Continuous Integration or Continuous Delivery (CI/CD) environment for nearly any combination of languages and source code repositories, as well as to automate other normal development operations. Jenkins' plugin design allows it to be expanded, allowing practically limitless possibilities for what Jenkins can achieve.

Jenkins' online interface, which includes on-the-fly error checks and built-in assistance, makes it simple to set up and configure. Jenkins connects with virtually every tool in the continuous integration and continuous delivery toolchain, thanks to hundreds of plugins in the Update Center.

To summarize, Jenkins is a strong automation tool that enables continuous integration and continuous delivery of projects independent of platform. It supports any type of build or continuous integration and is compatible with a variety of testing and deployment systems. Organizations may use Jenkins to automate the software development process and include development life-cycle operations such as build, document, test, package, stage, deploy static analysis, and much more. Jenkins uses plugins to perform Continuous Integration and can be simply set up and managed using its web interface, which features on-the-fly error checks and built-in assistance.

For the purposes of our thesis project, we have successfully installed and run Jenkins within a Docker container on a virtual machine (VM). This installation process was initiated by creating a YAML (YAML Ain't Markup Language) configuration file for Docker Compose, providing a structured and human-readable format for defining our Jenkins container's settings and dependencies.

A terminal window with a dark background and light text. The prompt is 'jenkins@jenkins:~/docker-jenkins\$'. The command 'cat docker-compose.yml' has been executed, and the output is a YAML configuration file. The file defines a service named 'jenkins' with the following properties: image: jenkins/jenkins:lts, privileged: true, user: root, ports: ['80:8080', '443:8443', '50000:50000'], container\_name: jenkins, and volumes: ['/home/jenkins/docker-jenkins/jenkins\_data:/var/jenkins\_home', '/var/run/docker.sock:/var/run/docker.sock']. The prompt returns to 'jenkins@jenkins:~/docker-jenkins\$' at the bottom.

```
jenkins@jenkins:~/docker-jenkins$ cat docker-compose.yml
version: '3.3'
services:
  jenkins:
    image: jenkins/jenkins:lts
    privileged: true
    user: root
    ports:
      - '80:8080'
      - '443:8443'
      - '50000:50000'
    container_name: jenkins
    volumes:
      - /home/jenkins/docker-jenkins/jenkins_data:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
jenkins@jenkins:~/docker-jenkins$
```

Figure 5: Jenkins compositionl file

Here's a breakdown of what this file does:

- version: '3.3': This indicates the version of the Docker Compose file format being used.
- services: This section defines a Docker service named "jenkins."

- `image: jenkins/jenkins:lts`: This specifies the Docker image to use for the Jenkins service. It uses the LTS (Long Term Support) version of the official Jenkins image from Docker Hub.
- `privileged: true`: This setting allows the Jenkins container to run in a privileged mode, which grants it additional privileges and access to the host system.
- `user: root`: It runs the Jenkins container with the root user. This is sometimes necessary when working with Docker-in-Docker (DinD) setups or when you need elevated privileges within the container.
- `ports`: This section defines port mappings, exposing ports from the Jenkins container to the host system. It maps:
  - Port 8080 in the container to port 80 on the host. Port 8443 in the container to port 443 on the host.
  - Port 50000 in the container to port 50000 on the host. Port 50000 is commonly used for Jenkins agent connections.
- `container_name: jenkins`: This sets a custom name for the Jenkins container.
- `volumes`: This section specifies volumes to be mounted inside the container. It maps:
  - `/home/jenkins/docker-jenkins/jenkins_data` on the host to `/var/jenkins_home` inside the container. This is where Jenkins stores its data, configurations, and plugins, allowing you to persist Jenkins data even if the container is destroyed.
  - `/var/run/docker.sock` on the host to `/var/run/docker.sock` inside the container. This allows the Jenkins container to interact with the Docker daemon on the host, enabling Jenkins to create and manage other Docker containers (useful for Jenkins pipelines that involve Docker-based builds).

After configuring our YAML file to define the Jenkins Docker container and its associated settings, we executed the command `docker-compose up -d` within the same directory as the YAML file. This command initiated the deployment of the Jenkins container, and as a result, Jenkins is now up and running on our system.



Welcome to Jenkins!


 Keep me signed in

Figure 6: Jenkins login screen

The pipelines implemented in this Jenkins are going to be described later in this thesis.

# OWASP WebGoat

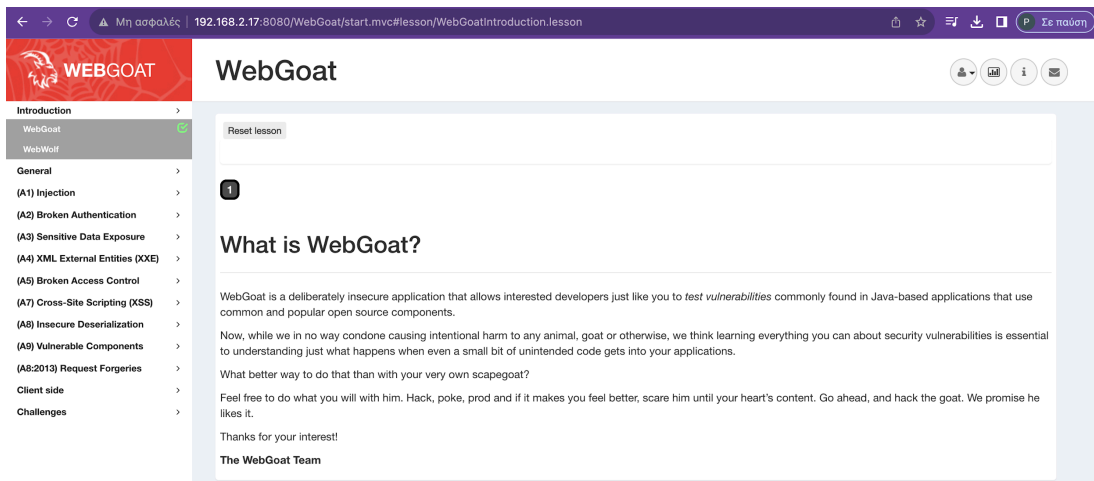


Figure 7: OWASP WebGoat User Interface

OWASP created WebGoat, a purposefully unsafe online application. It is intended to assist developers and security experts in understanding and preventing common web application vulnerabilities. WebGoat offers a hands-on learning environment in which users may engage with numerous susceptible elements and practice exploiting and correcting security problems.

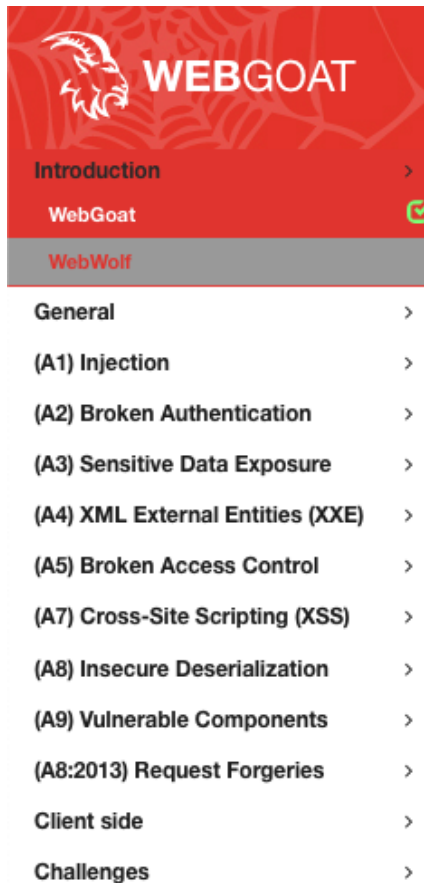


Figure 8: OWASP WebGoat Menu

Individuals get the opportunity to gain practical knowledge and skills in identifying and taking advantage of security weaknesses, including but not limited to cross-site scripting (XSS), SQL injection, authentication bypass, insecure session management, and several others, through the utilization of WebGoat. The curriculum offers a diverse range of courses and tasks that specifically target a multitude of security vulnerabilities commonly seen in online applications.

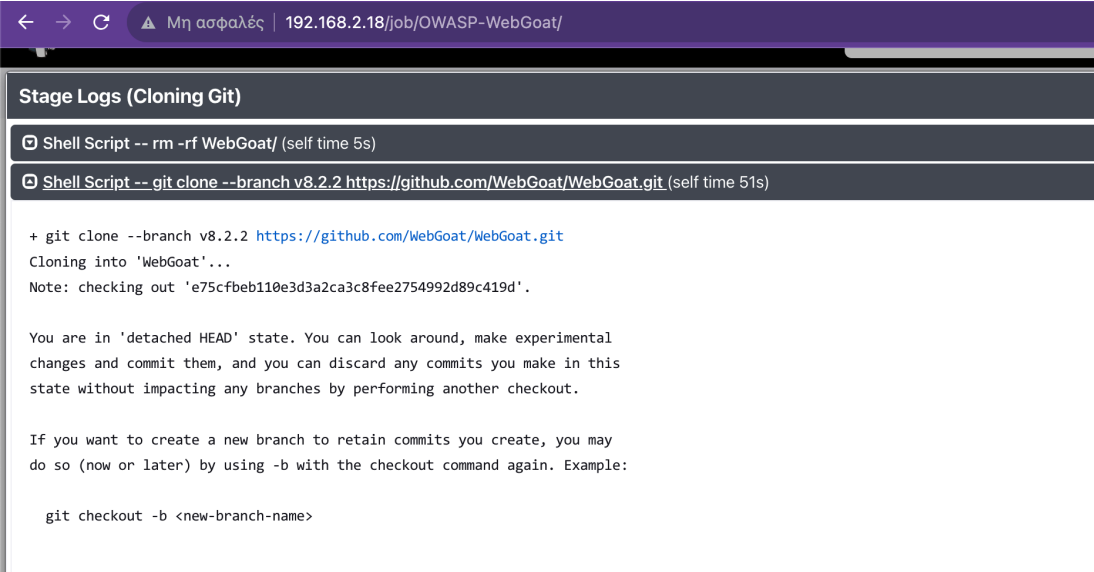
WebGoat is a freely available open-source initiative that may be obtained and implemented on personal computing devices. The software is implemented using the Java programming language and functions on a web server, so enabling accessibility through a web browser. The software application has a user-friendly interface that facilitates users in navigating the various courses, accessing explanatory materials, and engaging with the assigned assignments.

It is important to acknowledge that WebGoat is designed exclusively for educational purposes and should be employed just in limited scenarios or by those possessing legitimate authorization. The major objective of this initiative is to augment one's understanding of web application security and promote the use of secure coding methods.

**WARNING 1:** While running this program your machine will be extremely vulnerable to attack. You should disconnect from the Internet while using this program. WebGoat's default configuration binds to localhost to minimize the exposure.

**WARNING 2:** This program is for educational purposes only. If you attempt these techniques without authorization, you are very likely to get caught. If you are caught engaging in unauthorized hacking, most companies will fire you. Claiming that you were doing security research will not work as that is the first thing that all hackers claim.

In the context of this thesis, WebGoat has been employed as a subject for comprehensive testing through the implemented pipelines. As the initial step within the pipeline, the WebGoat Application code repository is replicated into our Jenkins workspace, situated on VM1 - the Jenkins Machine.



```
← → ↻ Μη ασφαλές | 192.168.2.18/job/OWASP-WebGoat/
Stage Logs (Cloning Git)
Shell Script -- rm -rf WebGoat/ (self time 5s)
Shell Script -- git clone --branch v8.2.2 https://github.com/WebGoat/WebGoat.git (self time 51s)
+ git clone --branch v8.2.2 https://github.com/WebGoat/WebGoat.git
Cloning into 'WebGoat'...
Note: checking out 'e75cfbeb110e3d3a2ca3c8fee2754992d89c419d'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

git checkout -b <new-branch-name>
```

Figure 9: OWASP WebGoat cloned via the pipeline

The primary objective of this process is to subject the WebGoat Application to rigorous scrutiny through Static Code Analysis, utilizing a suite of Static Application Security Testing (SAST) tools. Ultimately, the results obtained from these analyses will be subjected to a comparative evaluation.

Furthermore, as part of our comparative analysis of tools for Docker image scanning, we intend to incorporate the WebGoat Docker container in subsequent pipeline stages. This endeavour will facilitate an assessment of various Docker image scanning tools to enhance the overall security posture of the WebGoat Application.

### SAST using SNYK

“Snyk is a developer security platform. Integrating directly into development tools, workflows, and automation pipelines, Snyk makes it easy for teams to find, prioritize, and fix security vulnerabilities in code, dependencies, containers, and infrastructure as code. Supported by industry-leading application and security intelligence, Snyk puts security expertise in any developer’s toolkit.” [24]

Snyk embarked on its journey with a bold mission: to fundamentally change the way we approach software security. Through unwavering dedication to this goal, Snyk has emerged as a pioneer in the industry, reshaping how businesses view and manage vulnerabilities within their software applications and code repositories.

At the core of Snyk's acclaim lies its extensive array of features, strategically crafted to address security challenges from all perspectives. These capabilities encompass:

- **Vulnerability Scanning and Analysis:** Snyk employs cutting-edge techniques, including static and dynamic analysis, to identify vulnerabilities in code, dependencies, and container images. This holistic approach enables organizations to detect and mitigate risks at every layer of their software stack.
- **Dependency Management:** Snyk is renowned for its prowess in managing open-source dependencies. It scans and monitors third-party libraries and components, empowering developers to make informed choices and remediate vulnerabilities seamlessly.
- **DevSecOps Integration:** Snyk understands the pivotal role of security in the DevOps and DevSecOps paradigms. It seamlessly integrates into development pipelines, facilitating the early detection and remediation of vulnerabilities. This not only accelerates development but also bolsters security practices.

For the specific diploma thesis, the Snyk tool played a pivotal role in conducting a comprehensive static code analysis on the codebase of the WebGoat repository. In order to streamline this process, we implemented an autonomous and parallel pipeline, which seamlessly integrates with the primary pipeline responsible for the static code analysis, leveraging the capabilities of Snyk.

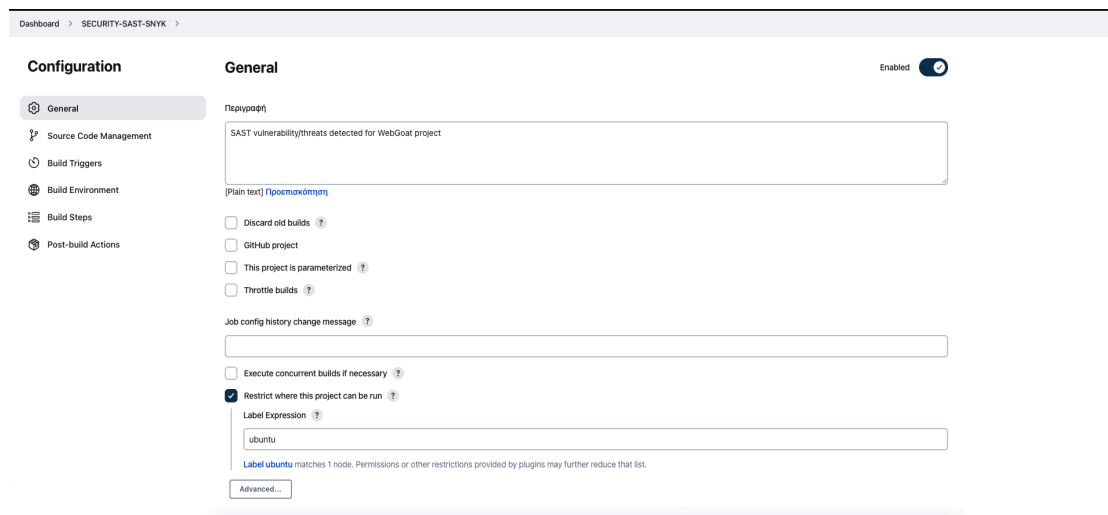


Figure 10: SAST SNYK Jenkins pipeline

To enable the execution of the Snyk tool within the pipeline, we deployed the Snyk Security plugin within our Jenkins environment. This plugin serves as the bridge, allowing us to establish a direct connection with the Snyk tool using a straightforward authentication token. This authentication token is provided by Snyk itself upon creating an account with the platform. By harnessing this token, the pipeline commences by performing a checkout of the WebGoat code from the Git repository,

thereby ensuring that it is working with the latest version of the codebase. Subsequently, it proceeds to undertake a thorough and in-depth static analysis of the specified codebase.

### Plugin Manager

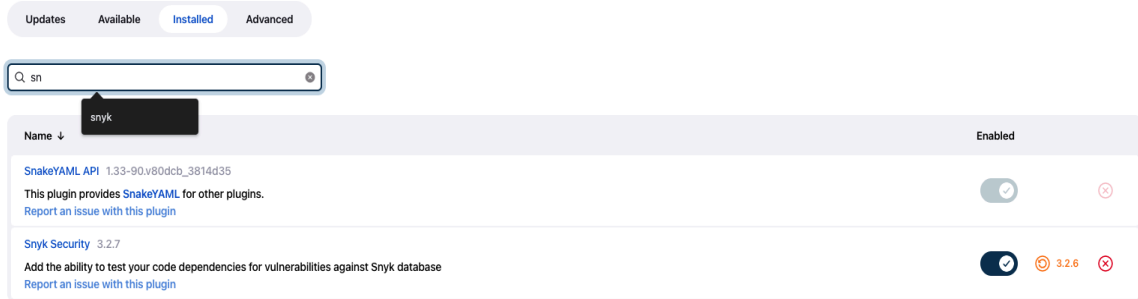


Figure 11: SNYK Plugin

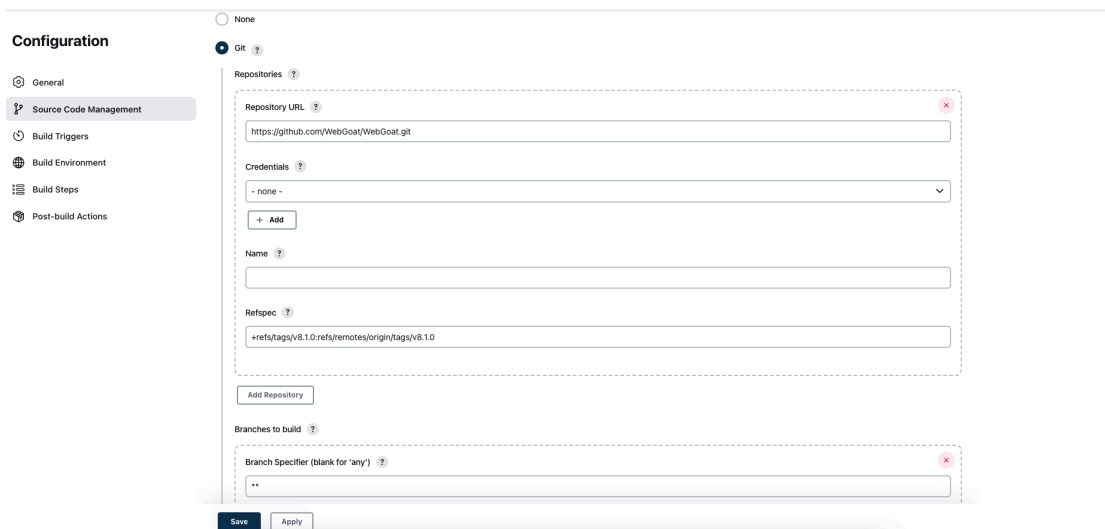


Figure 12: SAST SNYK Jenkins pipeline configuration 1

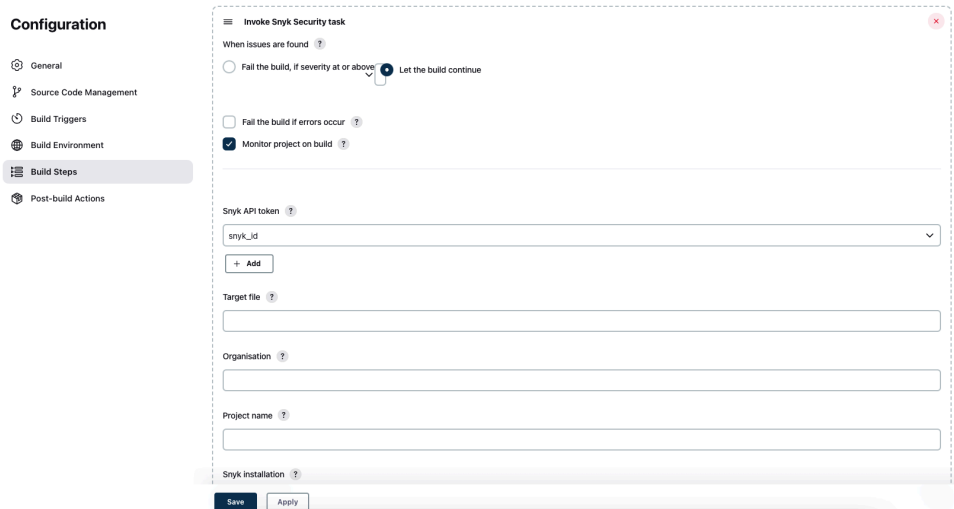


Figure 13: SAST SNYK Jenkins pipeline configuration 2

Upon the completion of the code analysis phase, we've configured the pipeline to generate a comprehensive report. This report serves as a valuable document, encapsulating vital information such as the timestamp of its creation, the specific path where the analysis was conducted, and, of course, a detailed account of the identified findings.

Post-build Actions

Publish HTML reports ?

Reports

HTML directory to archive ?

Index page[s] ?

Index page title[s] (Optional) ?

Report title ?

Publishing options...

Add

Add post-build action

Figure 14: SAST SNYK Jenkins pipeline configuration 3

Jenkins

Dashboard > SECURITY-SAST-SNYK > #32

↑ Επιστροφή στο Project

☰ Κατάσταση

</> Αλλαγές

📄 Console Output

📄 View as plain text

📄 Επεξεργασία Πληροφοριών Build

🗑️ Delete build '#32'

📄 Polling Log

🔗 Git Build Data

📄 Snyk Security Report

← Προηγούμενο Build

✅ **Build #32 (10 Σεπ 2023, 2:13:50 μ.μ.)**

📦 Build Artifacts

📄 2023-09-10T14-18-29-333898Z\_snyk\_report.html 80.14 KB view

</> No changes.

🕒 Started by an SCM change

🔗 Revision: e47f99439585519feea1032e6a00c129d0595834

🔗 Repository: <https://github.com/WebGoat/WebGoat.git>

- refs/tags/vtest2

Figure 15: SAST SNYK Security Report in Jenkins Pipeline Menu



The report is meticulously structured, categorizing the vulnerabilities discovered into distinct severity levels. The initial section is dedicated to critical vulnerabilities, followed by high, medium, and ultimately, low-severity vulnerabilities. This clear delineation allows for a prioritized approach to addressing potential issues within the codebase.

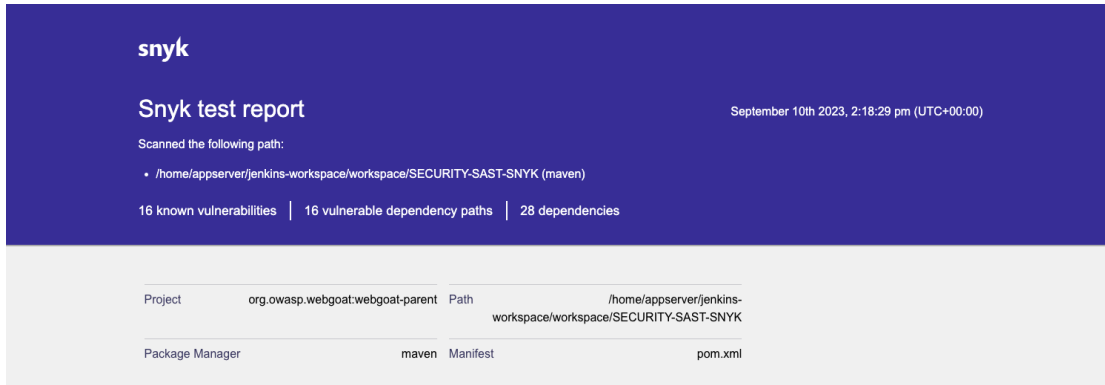


Figure 16: SAST SNYK Security Report Header

For each identified vulnerability, in addition to its severity level, we provide a wealth of information, including details about the package and module in which it was located, the precise path within the module where the issue was identified, a comprehensive description of the vulnerability itself, and a succinct overview of its significance. Furthermore, we include information on when the vulnerability was initially detected, the recommended remediation steps, and any other references or mentions made regarding the vulnerability.

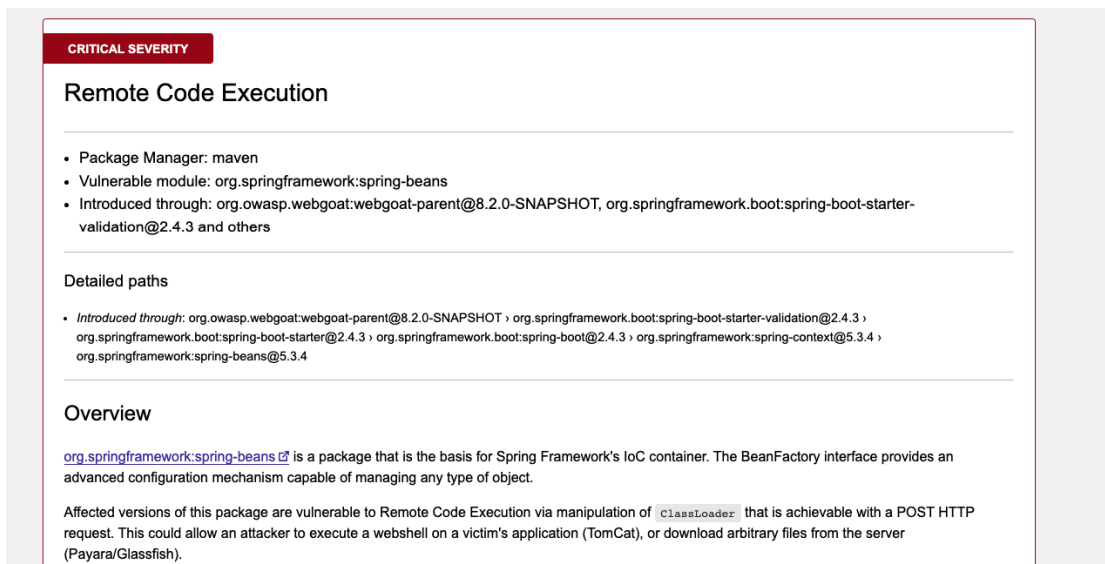


Figure 17: SAST SNYK Security Report Vulnerabilities

Finally, for those seeking to delve deeper into the intricacies of a specific vulnerability, we facilitate easy access to further information. A convenient link to additional details is thoughtfully provided at the conclusion of each section dedicated

to an individual vulnerability. This approach ensures that stakeholders have access to all the necessary information needed to understand, address, and mitigate potential security risks within the codebase effectively.

**Remediation**

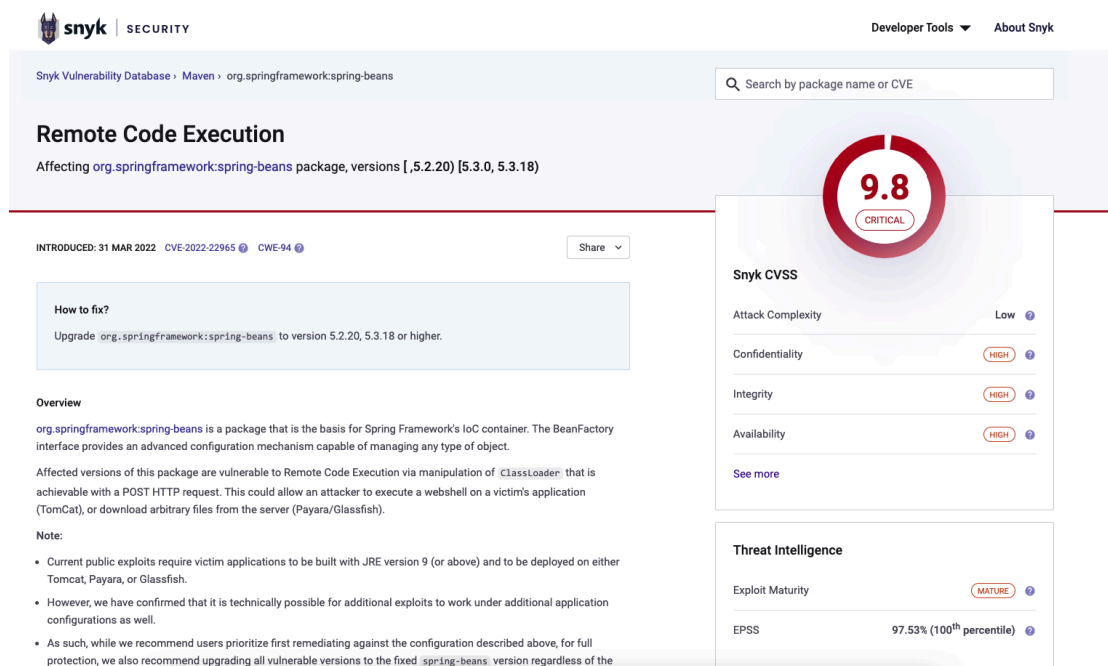
Upgrade `org.springframework:spring-beans` to version 5.2.20, 5.3.18 or higher.

**References**

- [CyberKendra Post](#)
- [GitHub Commit](#)
- [LunaSec Blog](#)
- [Payara Blogpost](#)
- [Payara PoC](#)
- [Snyk Blog - Technical Breakdown](#)
- [Snyk PoC](#)
- [Spring Security Announcement](#)
- [Spring Tomcat Mitigation Advice](#)
- [CISA - Known Exploited Vulnerabilities](#)
- [Nuclei Templates](#)

[More about this vulnerability](#)

Figure 18: SAST SNYK Security Report Vulnerabilities Remediation 1



**Remote Code Execution**  
Affecting `org.springframework:spring-beans` package, versions [ ,5.2.20) [5.3.0, 5.3.18)

**Snyk CVSS**  
9.8 (CRITICAL)

**Attack Complexity** Low

**Confidentiality** HIGH

**Integrity** HIGH

**Availability** HIGH

**Threat Intelligence**  
Exploit Maturity: MATURE  
EPSS: 97.53% (100<sup>th</sup> percentile)

**How to fix?**  
Upgrade `org.springframework:spring-beans` to version 5.2.20, 5.3.18 or higher.

**Overview**  
`org.springframework:spring-beans` is a package that is the basis for Spring Framework's IoC container. The BeanFactory interface provides an advanced configuration mechanism capable of managing any type of object. Affected versions of this package are vulnerable to Remote Code Execution via manipulation of `ClassLoader` that is achievable with a POST HTTP request. This could allow an attacker to execute a webshell on a victim's application (Tomcat), or download arbitrary files from the server (Payara/Glassfish).

**Note:**

- Current public exploits require victim applications to be built with JRE version 9 (or above) and to be deployed on either Tomcat, Payara, or Glassfish.
- However, we have confirmed that it is technically possible for additional exploits to work under additional application configurations as well.
- As such, while we recommend users prioritize first remediating against the configuration described above, for full protection, we also recommend upgrading all vulnerable versions to the fixed `spring-beans` version regardless of the

Figure 19: SAST SNYK Security Report Vulnerabilities Remediation 2

In essence, the integration of the Snyk tool and the meticulous pipeline configuration have not only streamlined the static code analysis process but have also facilitated a comprehensive reporting system that empowers our team to make informed decisions regarding code quality and security.

Overall, in the table below is described what the Snyk tool typically provides as part of its static code analysis results:

Attribute	Description
Vulnerability ID	A unique identifier for the discovered vulnerability.
Severity	The level of severity assigned to the vulnerability (e.g., critical, high, medium, low).
Package	The software package or library containing the vulnerability.
Module	The specific module or component within the package where the vulnerability was found.
Path	The detailed file path or location within the module where the vulnerability exists.
Description	A textual description of the vulnerability, providing context and details about the issue.
Remediation	Recommendations or steps to address and fix the vulnerability.
References	Any additional references, links, or resources related to the vulnerability

*Table 1: Snyk tool results description*

Furthermore, the configuration process using the plugin proved to be straightforward, and the exporting of the report within Jenkins was highly facilitative. Additionally, the scanning process itself demonstrated remarkable efficiency, completing in a mere 5 minutes.

The seamless integration of the SNYK tool, made possible through the user-friendly plugin, greatly expedited the setup and execution of the static code analysis. This efficiency not only saved valuable time but also enhanced the overall productivity of the development workflow. Incorporating such tools into our development pipeline not only contributes to improved code quality and security but also streamlines the development process itself.

This, in turn, allows a team to focus more on proactive development and addressing identified vulnerabilities swiftly. As we continually refine our development practices, the integration of tools like SNYK remains an essential component of our commitment to delivering secure and high-quality software solutions. The rapid scan

times and ease of configuration provided by the SNYK tool, as well as its seamless integration with Jenkins, have proven to be invaluable assets in our pursuit of software excellence.

Here's a table summarizing both the positive and negative aspects of using the SNYK tool for static code analysis:

Aspect	Positive	Negative
Ease of Integration	- Seamless integration with Jenkins using the Snyk Security plugin.	- Initial setup might require some configuration, but this is a one-time effort.
Efficiency	- Rapid code analysis, completing scans in a short time (e.g., 5 minutes).	- Efficiency may vary depending on the size and complexity of the codebase.
Security Insights	- Provides valuable security insights by identifying vulnerabilities.	- Some false positives or negatives may occur, requiring manual review.
Severity Classification	- Categorizes vulnerabilities by severity (e.g., critical, high, medium).	- Severity classification may not always align perfectly with specific project needs.
Detailed Reporting	- Generates comprehensive reports with detailed information about vulnerabilities.	- Reports can be lengthy and may require thorough analysis to prioritize and address issues.
Authentication	- Easy authentication using a provided token for direct integration with SNYK.	- Token management and security are essential to prevent unauthorized access.
Ease of Use	- User-friendly interface for configuring and managing scans.	- Configuration may require some familiarity with Jenkins and SNYK settings.
Integration Flexibility	- Integrates with various CI/CD pipelines, enhancing DevSecOps practices.	- Compatibility with other CI/CD tools might require additional configuration or scripting.
Continuous Monitoring	- Supports ongoing security monitoring with scheduled scans.	- Frequent scans may generate a high volume of reports, necessitating effective management.
GitHub Compatibility	- Compatible with GitHub repositories, a widely used platform for source code hosting.	- In case of GitHub SSH issues, additional troubleshooting may be necessary.
Learning Curve	- Relatively straightforward for those familiar with CI/CD and security practices.	- New users may require time to learn the tool's capabilities and nuances.
Community and Support	- Active community and support resources for troubleshooting and guidance.	- Support may vary, and resolution times for issues may depend on the level of service chosen.

Aspect	Positive	Negative
Cost	- Offers both free and paid plans, allowing flexibility based on project requirements.	- Costs may increase for large-scale or complex projects with additional features or services.

*Table 2: Snyk tool positives and negatives*

## SAST using SonarQube

SonarQube, previously known as "Sonar," was initially developed by SonarSource in 2007 as an open-source platform for continuous inspection of code quality [25]. It has since evolved to encompass a wider range of software quality management, including security analysis [25]. SonarQube gained significant recognition and a dedicated user community, establishing SonarSource as a prominent player in the software quality and security field [25].

Some key points about SonarQube:

- SonarQube is an open-source platform for continuous inspection of code quality [25].
- It performs automatic reviews with static analysis of code to detect bugs, code smells, and security vulnerabilities across multiple programming languages [25].
- SonarQube offers reports on various aspects of code quality, including duplicated code, coding standards, unit tests, code coverage, code complexity, comments, bugs, and security recommendations [25].
- It provides metrics history and evolution graphs to track the progress of code quality over time [25].
- SonarQube integrates with popular build tools like Maven, Ant, Gradle, and MSBuild, as well as continuous integration tools such as Jenkins and Bamboo [25].
- It supports a wide range of programming languages, including Java, C#, C, C++, JavaScript, Python, Go, Swift, PHP, and many more [25].
- SonarQube can be extended with plugins and integrates with development environments like Eclipse, Visual Studio, Visual Studio Code, and IntelliJ IDEA through SonarLint plugins [25].
- It offers both a free community edition under the GNU Lesser General Public License and commercial editions with additional features and support [25].
- SonarQube has a concept of quality gates that define the criteria for code quality. A passing quality gate indicates that the code meets the defined standards, while a failing quality gate indicates issues that need to be addressed [26].
- SonarQube provides feedback through its user interface, email notifications, and decorations on pull or merge requests (in commercial editions) [26].
- Developers can obtain in-depth guidance on identified issues, including why each issue is a problem and how to fix it [26].

Overall, SonarQube is a comprehensive code quality and security analysis tool that helps developers maintain clean, reliable, and maintainable code throughout the

development process [26]. It offers a wide range of features, supports multiple programming languages, and integrates with popular development tools and environments [25] [26].

The SonarQube application has been successfully deployed on VM2, also known as 'appserver,' using a Docker container. It is currently operational and accessible through port 9000. To facilitate static code analysis, we have integrated Jenkins with SonarQube on VM2, employing a dedicated Jenkins plugin called 'SonarQube Scanner.' This integration allows Jenkins to establish a connection with SonarQube, which is running on port 9000 on VM2. Through this connection, Jenkins can transmit the source code to SonarQube for comprehensive static analysis.

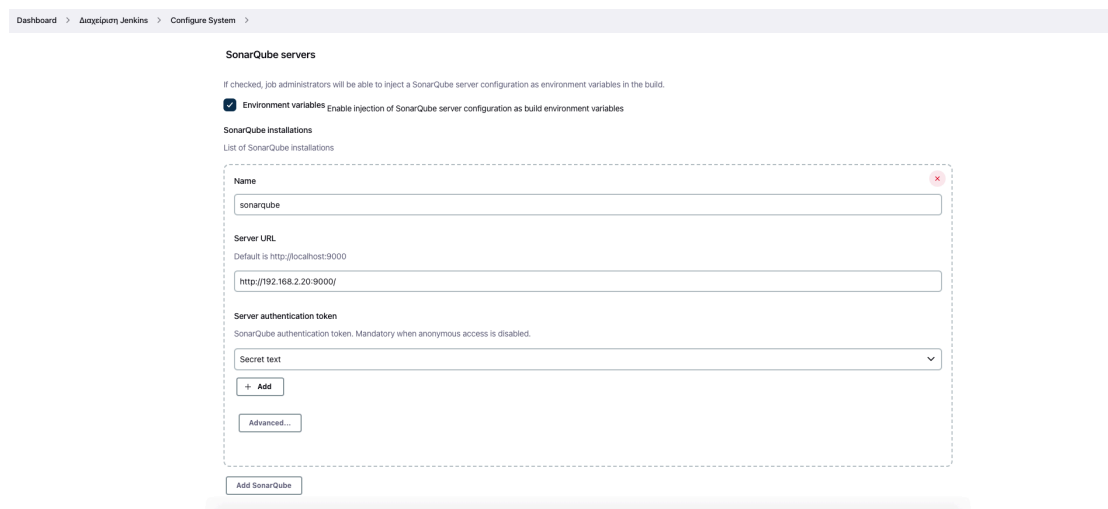


Figure 20: SAST SonarQube Jenkins pipeline

As evident from the previous image, it is necessary to declare the SonarQube server, which is the server running within a Docker container on VM2, as an environment variable in the "Configure System" page of Jenkins. Specifically, the following information needs to be specified: the server's name and the URL at which it is accessible. In order to ensure seamless integration and effective communication between Jenkins and SonarQube, these server details must be accurately provided within the Jenkins configuration. This step is crucial for enabling Jenkins to connect with and utilize the SonarQube server for code analysis and related tasks.

Subsequently, a parallel pipeline was created using the core pipeline. This parallel pipeline is responsible for sending the code of the WebGoat application to the SonarQube server for static analysis. In this workflow, the core pipeline is effectively orchestrating the code analysis process by coordinating the transfer of code to the SonarQube server. This ensures that the code undergoes comprehensive static analysis, helping to identify and address potential issues and vulnerabilities in the WebGoat application. This parallel approach to code analysis can significantly enhance code quality and security within the project.



Figure 21: SAST SonarQube Jenkins pipeline configuration

In order to achieve this, as depicted in the image, it was necessary to declare the properties within the configuration of the pipeline. These properties are required by SonarQube to perform static code analysis. These properties typically include:

- `sonar.projectKey`: This is a unique identifier for your project in SonarQube. It helps distinguish different projects within SonarQube.
- `sonar.projectName`: This property specifies the name of your project in SonarQube.
- `sonar.projectVersion`: It indicates the version of your project. You can use this to track changes or versions of your project in SonarQube.
- `sonar.language`: This property specifies the primary programming language used in your project. In this case, it's set to Java.
- `sonar.exclusions`: This property is used to specify file or directory patterns that should be excluded from the SonarQube analysis. In your configuration, it excludes all TypeScript files (.ts files).
- `sonar.login` and `sonar.password`: These properties are used for authentication when connecting to SonarQube. In your configuration, the login is set to "admin," but the password is empty, which means you likely need to provide the actual password for authentication.
- `sonar.projectBaseDir`: This property sets the base directory for your project. It's the root directory where SonarQube should start its analysis.
- `sonar.java.binaries`: This property should specify the location of the compiled Java binaries (class files) of your project. This is necessary for SonarQube to analyze your Java code.

Furthermore, the "-X" argument was added to the additional arguments. This was done to enable detailed logging in the Jenkins output. This allows anyone reviewing the Jenkins output to access comprehensive log information.

After the pipeline was successfully executed, one can observe the creation of a SonarQube report. This report provides a detailed analysis of the codebase, highlighting issues, code quality metrics, and security vulnerabilities. It serves as a

valuable resource for developers and teams to understand the state of their codebase and take appropriate actions to improve code quality and security.

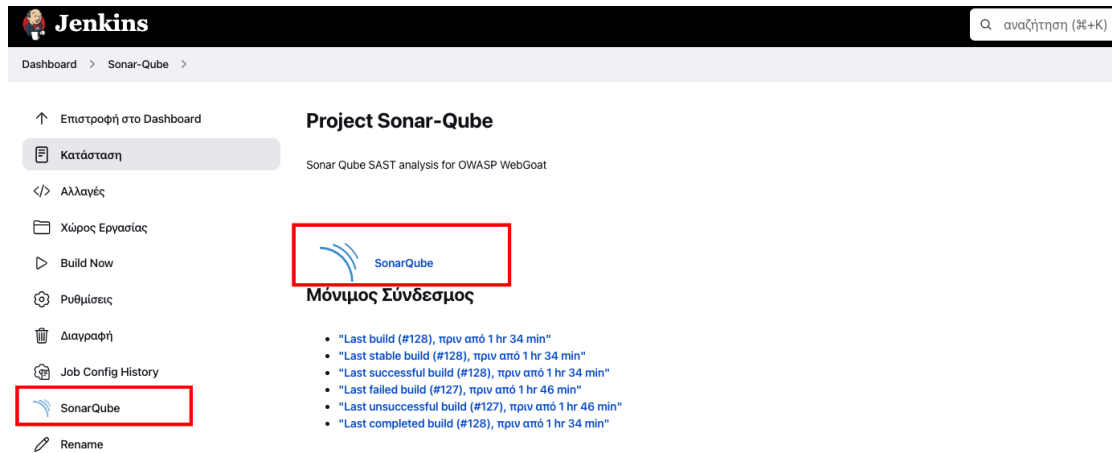


Figure 22: SAST SonarQube Security Report in Jenkins Pipeline Menu

This specific SonarQube report serves as a hyperlink that directs users to the SonarQube server's webpage, displaying the results of the particular code analysis. As depicted in the image, SonarQube provides a comprehensive breakdown of what's wrong with the project's code. Depending on the rules and criteria set by the user, the project can either pass or fail the analysis.

This detailed analysis report is a valuable resource for developers and teams to gain insights into code quality issues, security vulnerabilities, and compliance with coding standards. It allows for informed decision-making and facilitates the process of improving and maintaining the project's codebase.

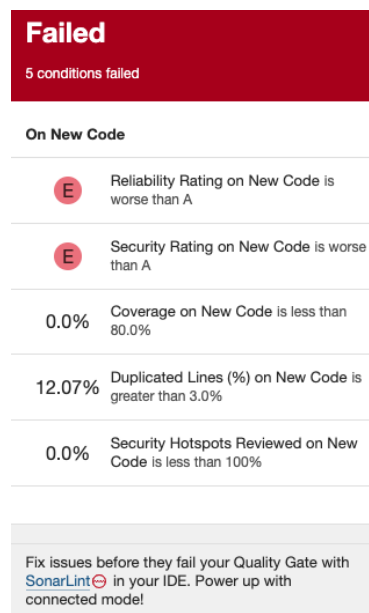


Figure 23: SAST SonarQube Project Status



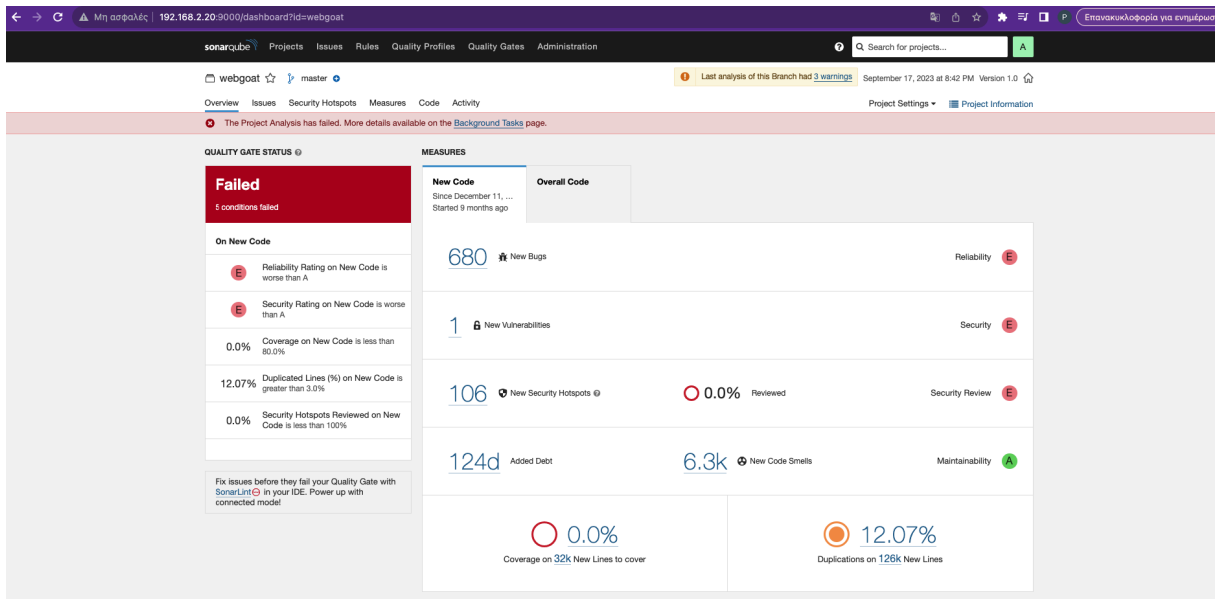


Figure 24: SAST SonarQube Project Vulnerabilities and Issues of new Code

In the scenario where builds occur continuously in a CI/CD environment, SonarQube has the capability to maintain statistics and also view metrics specifically for the new code. SonarQube's ability to retain statistics and focus on metrics for the new code is instrumental in ensuring code quality and security throughout the development lifecycle. This feature allows development teams to continuously monitor and assess the impact of changes introduced with each build. It facilitates the identification of issues and the enforcement of coding standards, ultimately contributing to the ongoing improvement of code quality and the reduction of technical debt. By concentrating on the new code, teams can efficiently prioritize their efforts and address code quality concerns in a streamlined manner.

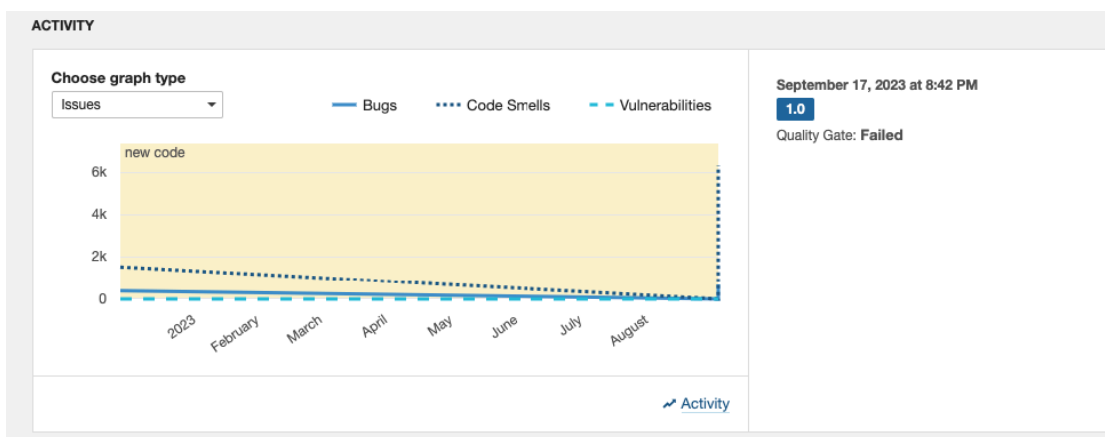


Figure 25: SAST SonarQube Project Vulnerabilities and Issues metrics

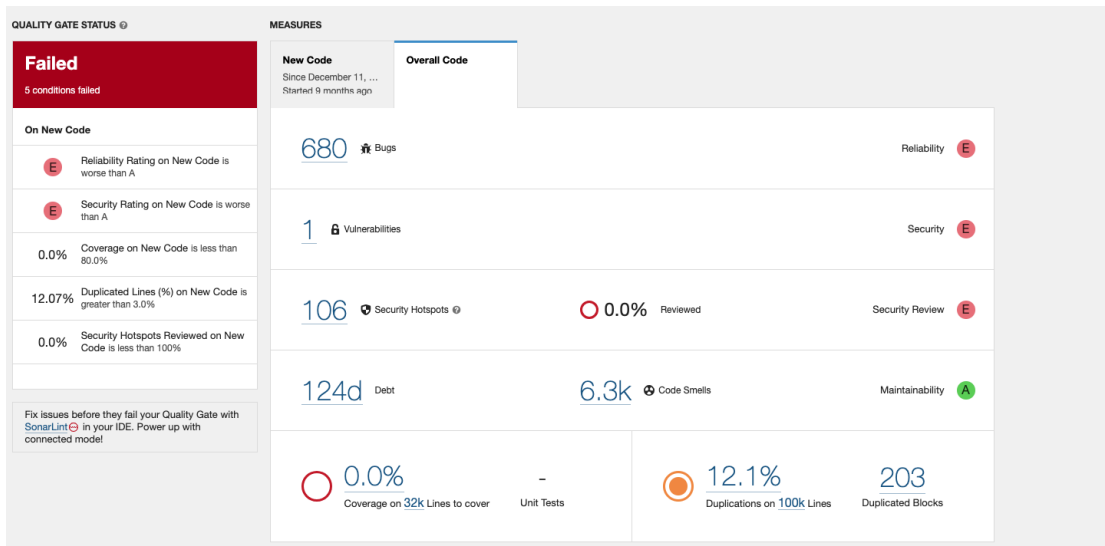


Figure 26: SAST SonarQube Project Vulnerabilities and Issues of Overall Code

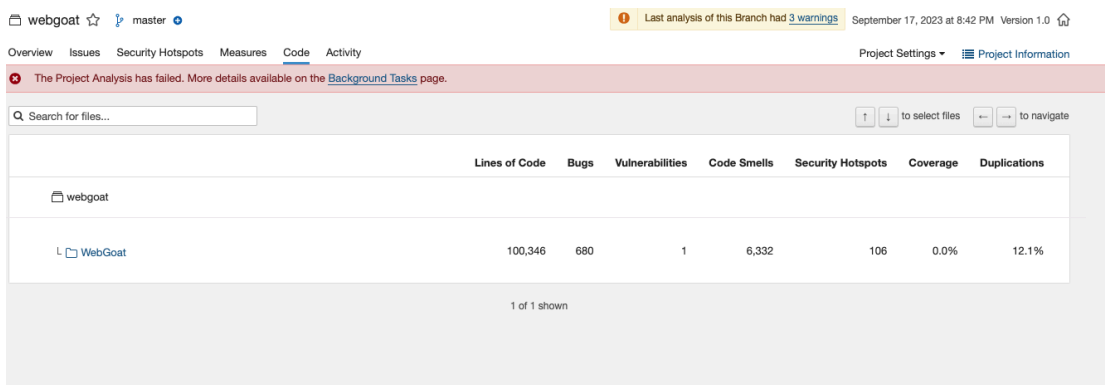


Figure 27: SAST SonarQube Project Source Code

The configuration for establishing a connection between Jenkins and SonarQube was straightforward and well-documented on both SonarQube and Jenkins web pages. It's important to highlight that SonarQube provides highly detailed reports, enabling the development team to thoroughly review code issues and propose improvements through a centralized platform. This centralized platform fosters collaboration among team members, allowing them to engage in discussions, share insights, and collectively work towards enhancing code quality and security. With access to comprehensive reports and a unified environment for code analysis, the team can make informed decisions and continuously strive to improve the overall quality of their codebase.

SonarQube is a valuable tool for code quality analysis, it's important to be aware of potential negatives or challenges that I met during the configuration and reviewing the specific tool:

- **False Positives:** SonarQube may sometimes report issues that are not actual problems in the code, leading to false positives. Developers may waste time investigating and fixing non-issues.

- **Overwhelming Feedback:** If not properly configured, SonarQube can generate many issues, overwhelming developers with too much feedback to address. This can lead to frustration and decreased productivity.
- **Performance Impact:** Running SonarQube analysis on large codebases can be time-consuming and resource-intensive, potentially slowing down the development process. For WebGoat analysis it was needed at least 30 minutes.
- **Learning Curve:** It is needed a lot of time to learn how to interpret SonarQube reports and understand the various rules and metrics it uses, which can be a learning curve.

This table summarizes both the positive aspects and considerations when using SonarQube for code quality analysis and static code analysis.

Aspect	Positive Aspects	Negative Aspects
Code Quality Analysis	Provides comprehensive code quality analysis with a focus on maintainability, reliability, and efficiency.	Some complex code patterns may lead to false positives or false negatives in the analysis.
Security Analysis	Offers security analysis for common vulnerabilities such as OWASP Top 10, ensuring safer code practices.	May not cover all security vulnerabilities, and additional security tools may be needed.
Custom Rules	Supports custom code quality and security rules creation, allowing organizations to tailor checks.	Creating and maintaining custom rules can be time-consuming and require expertise.
Integration	Integrates seamlessly with various development environments, CI/CD pipelines, and IDEs.	Setting up and configuring integrations can be complex and time-consuming.
Interactive Dashboards	Offers interactive and customizable dashboards for code quality and security metrics.	Navigating the UI and interpreting complex reports may require training for some users.
Scalability	Scales well to analyze large and complex codebases.	Requires sufficient hardware resources for large codebases and analysis configurations.
**Support for Multiple Languages**	Supports multiple programming languages, including Java, JavaScript, Python, C#, and more.	Analysis results and features may vary in depth and support across different programming languages.
Continuous Monitoring	Provides continuous code quality and security monitoring to catch issues early in the development	Setting up and maintaining continuous monitoring workflows can be challenging.

Aspect	Positive Aspects	Negative Aspects
Community and Ecosystem	Benefits from an active community and ecosystem with plugins, extensions, and support.	Some advanced features may require a paid license.
Cost (Community Edition)	Offers a free Community Edition with basic features	Advanced features and support require a paid license.

*Table 3: SonarQube tool positives and negatives*

## Container Security using Trivy

Trivy is an open-source security tool developed by Aqua Security. It is a comprehensive and versatile security scanner that focuses on vulnerability scanning in containerized environments [1], [27]. Trivy stands for "Tri" as in triple and "vy" as in vulnerability [1]. It has gained significant attention and adoption within the industry due to its reliability, frequent updates, and comprehensive vulnerability scanning capabilities [1].

Trivy is a comprehensive and versatile security scanner that originated from the security-conscious landscape of containerization, where Docker and other container technologies were gaining popularity for packaging and deploying applications [27]. Aqua Security, a cybersecurity firm specializing in container security solutions, recognized the potential security challenges in containerized environments and initiated the development of Trivy [27]. Trivy was first made available to the public in open-source form, aligning with the industry's inclination towards collaborative security practices [27].

Trivy is a specialized vulnerability scanner that is tailored explicitly for container images and filesystems. It is designed to address security concerns that arise when using containers [28]. Trivy scans container images, including their components such as packages and libraries, to identify known vulnerabilities. By doing so, it equips developers, DevOps teams, and security professionals with the means to identify and remediate security weaknesses before they can be exploited [28].

### Features and Capabilities:

Trivy boasts a robust set of features and capabilities that contribute to its effectiveness as a container security scanner:

- Container Image Scanning:** Trivy excels in scanning container images, encompassing both base images and application images. It inspects every layer of the image, assessing each component for vulnerabilities.
- Filesystem Scanning:** In addition to image scanning, Trivy can analyze the filesystems within containers. This capability is particularly valuable for identifying vulnerabilities within running containers, where traditional image scanning may not suffice.
- Image Format Support:** Trivy exhibits versatility by offering support for a variety of container image formats, including Docker images and OCI (Open Container Initiative) images. This compatibility ensures its applicability across different containerization technologies.

### Vulnerability Database:

One of Trivy's foundational strengths lies in its reliance on an extensive and frequently updated vulnerability database. This database, curated and maintained by Aqua Security and the open-source community, plays a pivotal role in the tool's ability to identify security flaws accurately.

For the specific thesis project, Trivy was employed to conduct a vulnerability assessment on the WebGoat container. The objective of this endeavor was to systematically evaluate the security posture of the WebGoat container by identifying potential vulnerabilities within its components. The methodology employed involved retrieving the WebGoat container from the Docker Hub repository and subjecting it to

a comprehensive security scan using the Trivy tool. This process was integrated into the overarching project workflow, wherein the central pipeline initiated the execution of a parallel pipeline, denoted as "Trivy." The "Trivy" pipeline was designed to orchestrate and facilitate the scanning process. Within this context, Trivy played a pivotal role in enhancing the security assurance of the WebGoat container, ensuring that any known vulnerabilities were diligently identified. This proactive approach aligns with contemporary best practices in container security, fortifying the overall resilience of containerized applications in a rapidly evolving cybersecurity landscape.

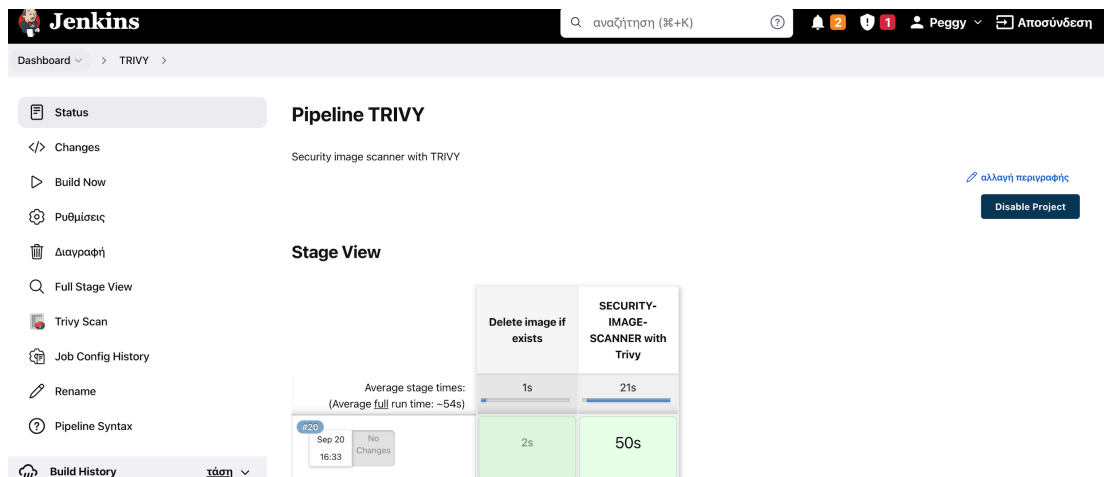


Figure 28: SCA Trivy Jenkins Pipeline

The pipeline has been internally configured within the Jenkins environment through the utilization of the following Groovy script:

```

Script ?
1 pipeline {
2   agent {
3     label 'ubuntu2'
4   }
5   stages {
6     stage('Delete image if exists') {
7       steps {
8         sh 'echo "Deletion stage"'
9       }
10    }
11    stage('SECURITY-IMAGE-SCANNER with Trivy'){
12      steps {
13        sh 'echo scan image for security'
14      }
15      // Install trivy
16      /*sh 'curl -sfl https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s --'
17      sh 'curl -sfl https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/html.tpl > html.tpl'
18
19      // Scan all vuln levels
20      sh 'mkdir -p reports'
21      sh 'trivy image webgoat/goatandwolf:latest --format template --template "@html.tpl" -o reports/trivy.html'
22      publishHTML target : [
23        allowMissing: true,
24        alwaysLinkToLastBuild: true,
25        keepAll: true,
26        reportDir: 'reports',
27        reportFiles: 'trivy.html',
28        reportName: 'Trivy Scan',
29        reportTitles: 'Trivy Scan'
30      ]
31    }
32  }
33 }
34

```

Figure 29: SCA Trivy Jenkins Pipeline configuration

As depicted in the image, Trivy is executed on the server labeled "appserver2" running the Ubuntu operating system (referred to as "ubuntu2"). The process involves several key steps:

- **Image Handling:** Trivy begins by evaluating the presence of a specified container image. If the image exists, Trivy initiates its removal to ensure a clean slate for subsequent operations.
- **Image Retrieval:** Subsequently, Trivy proceeds to download the designated container image anew from the Docker Hub repository. This step guarantees the use of the latest version of the image for the scanning process.
- **Template Download:** In preparation for the post-scanning phase, Trivy downloads an HTML template. This template serves as the foundation for generating a comprehensive report of the scanning results.
- **Scanning Process:** Trivy commences the scanning of the container image. The scanning process leverages the extensive vulnerability database maintained by Trivy to identify security vulnerabilities within the image's components.
- **Report Generation:** Upon completion of the scanning process, Trivy utilizes the downloaded HTML template to dynamically generate a detailed report summarizing the scanning results. This report is a reflection of the vulnerabilities discovered in the container image, sourced from Trivy's vulnerability database.
- **Integration with Jenkins:** The generated report is subsequently integrated into the Jenkins environment. This entails the publication of the report on the Jenkins platform, allowing for easy accessibility and review by relevant stakeholders.

As previously mentioned, the report detailing the identified vulnerabilities is made available on the Jenkins platform and is denoted as "Trivy Scan."

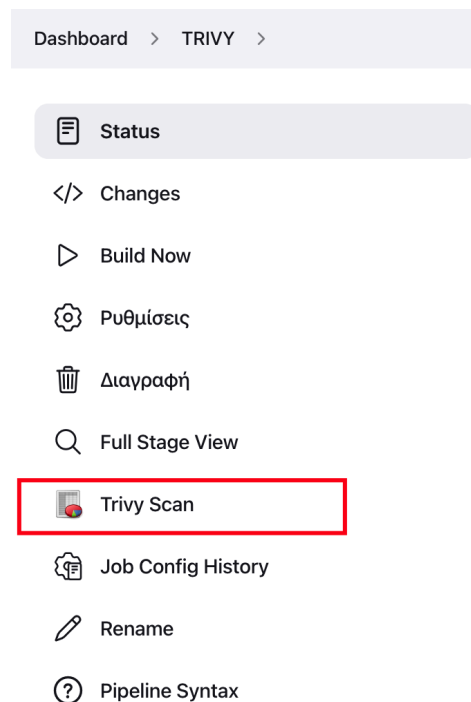


Figure 30: SCA Trivy Report in Jenkins Pipeline Menu

## webgoat/goatandwolf:latest (debian 11.0) - Trivy Report - 2023-09-20 16:34:27.421687288 +0000 UTC m=+43.955685865

Package	Vulnerability ID	Severity	Installed Version	Fixed Version	Links
apt	CVE-2011-3374	LOW	2.2.4		<a href="https://access.redhat.com/security/cve/cve-2011-3374">https://access.redhat.com/security/cve/cve-2011-3374</a> <a href="https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480">https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480</a> <a href="https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html">https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html</a> <a href="https://seclists.org/fulldisclosure/2011/09/01">https://seclists.org/fulldisclosure/2011/09/01</a> <a href="https://tracker.debian.org/tracker/CVE-2011-3374">https://tracker.debian.org/tracker/CVE-2011-3374</a> <a href="https://snyk.io/vuln/SNYK-LINUX-APT-116518">https://snyk.io/vuln/SNYK-LINUX-APT-116518</a> <a href="https://ubuntu.com/security/2011/09/01">https://ubuntu.com/security/2011/09/01</a>
apt-utils	CVE-2011-3374	LOW	2.2.4		<a href="https://access.redhat.com/security/cve/cve-2011-3374">https://access.redhat.com/security/cve/cve-2011-3374</a> <a href="https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480">https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=642480</a> <a href="https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html">https://people.canonical.com/~ubuntu-security/cve/2011/CVE-2011-3374.html</a> <a href="https://seclists.org/fulldisclosure/2011/09/01">https://seclists.org/fulldisclosure/2011/09/01</a> <a href="https://tracker.debian.org/tracker/CVE-2011-3374">https://tracker.debian.org/tracker/CVE-2011-3374</a> <a href="https://snyk.io/vuln/SNYK-LINUX-APT-116518">https://snyk.io/vuln/SNYK-LINUX-APT-116518</a> <a href="https://ubuntu.com/security/2011/09/01">https://ubuntu.com/security/2011/09/01</a>
bash	CVE-2022-3715	HIGH	5.1-2		<a href="https://access.redhat.com/errata/RHSA-2023-0340">https://access.redhat.com/errata/RHSA-2023-0340</a> <a href="https://access.redhat.com/security/cve/CVE-2022-3715">https://access.redhat.com/security/cve/CVE-2022-3715</a> <a href="https://bugzilla.redhat.com/show_bug.cgi?id=2126720">https://bugzilla.redhat.com/show_bug.cgi?id=2126720</a> <a href="https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-3715">https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-3715</a> <a href="https://errata.almalinux.org/9/ALSA-2023-0340.html">https://errata.almalinux.org/9/ALSA-2023-0340.html</a> <a href="https://errata.rockylinux.org/RLSA-2023-0340">https://errata.rockylinux.org/RLSA-2023-0340</a> <a href="https://linux.oracle.com/errata/ELSA-2023-0340.html">https://linux.oracle.com/errata/ELSA-2023-0340.html</a> <a href="https://lists.gnu.org/archive/html/bug-bash/2022-09/01">https://lists.gnu.org/archive/html/bug-bash/2022-09/01</a> <a href="https://nvd.nist.gov/vuln/detail/CVE-2022-3715">https://nvd.nist.gov/vuln/detail/CVE-2022-3715</a> <a href="https://www.cve.org/CVERecord?id=CVE-2022-3715">https://www.cve.org/CVERecord?id=CVE-2022-3715</a> <a href="http://packetstormsecurity.com/files/170176/snap-confine-must_mkdir_and_open_with_perms-Race-Condition.html">http://packetstormsecurity.com/files/170176/snap-confine-must_mkdir_and_open_with_perms-Race-Condition.html</a> <a href="http://seclists.org/fulldisclosure/2022/Dec/4">http://seclists.org/fulldisclosure/2022/Dec/4</a> <a href="http://www.openwall.com/lists/oss-security/2022/11/30/2">http://www.openwall.com/lists/oss-security/2022/11/30/2</a> <a href="https://access.redhat.com/security/cve/CVE-2022-3715">https://access.redhat.com/security/cve/CVE-2022-3715</a> <a href="https://bugzilla.redhat.com/show_bug.cgi?id=2024631">https://bugzilla.redhat.com/show_bug.cgi?id=2024631</a> <a href="https://github.com/utl-linxx/utl-linxx/commit/57202f5713afa2af20ffbb">https://github.com/utl-linxx/utl-linxx/commit/57202f5713afa2af20ffbb</a> <a href="https://mirrors.edge.kernel.org/pub/linux/utils/util-linux/v2.37/v2.37.3-ReleaseNotes">https://mirrors.edge.kernel.org/pub/linux/utils/util-linux/v2.37/v2.37.3-ReleaseNotes</a> <a href="https://nvd.nist.gov/vuln/detail/CVE-2022-3715">https://nvd.nist.gov/vuln/detail/CVE-2022-3715</a> <a href="https://security.netapp.com/advisory/ntap-20221209-0002/">https://security.netapp.com/advisory/ntap-20221209-0002/</a> <a href="https://ubuntu.com/security/notices/USN-5279-1">https://ubuntu.com/security/notices/USN-5279-1</a> <a href="https://www.openwall.com/lists/oss-security/2022/01/24/2">https://www.openwall.com/lists/oss-security/2022/01/24/2</a>
bsdutils	CVE-2021-3995	MEDIUM	2.36.1-8	2.36.1-8+deb11u1	<a href="https://access.redhat.com/errata/RHSA-2021-3995">https://access.redhat.com/errata/RHSA-2021-3995</a> <a href="https://github.com/utl-linxx/utl-linxx/commit/57202f5713afa2af20ffbb">https://github.com/utl-linxx/utl-linxx/commit/57202f5713afa2af20ffbb</a> <a href="https://mirrors.edge.kernel.org/pub/linux/utils/util-linux/v2.37/v2.37.3-ReleaseNotes">https://mirrors.edge.kernel.org/pub/linux/utils/util-linux/v2.37/v2.37.3-ReleaseNotes</a> <a href="https://nvd.nist.gov/vuln/detail/CVE-2021-3995">https://nvd.nist.gov/vuln/detail/CVE-2021-3995</a> <a href="https://security.netapp.com/advisory/ntap-20221209-0002/">https://security.netapp.com/advisory/ntap-20221209-0002/</a> <a href="https://ubuntu.com/security/notices/USN-5279-1">https://ubuntu.com/security/notices/USN-5279-1</a> <a href="https://www.openwall.com/lists/oss-security/2022/01/24/2">https://www.openwall.com/lists/oss-security/2022/01/24/2</a>

Figure 31: SCA Trivy Report

The "Trivy Scan" report is generated in HTML format and comprises several columns of essential information for comprehensive vulnerability assessment. These columns include:

1. **Package:** This column lists the specific software packages or libraries within the container image that have been assessed for vulnerabilities.
2. **Vulnerability ID:** Each vulnerability detected is associated with a unique identification code or Vulnerability ID. This facilitates precise tracking and reference for remediation efforts.
3. **Severity:** The severity level of each identified vulnerability is indicated, allowing stakeholders to prioritize remediation actions based on the potential impact of the vulnerability. Severity levels typically range from Critical to Low.
4. **Installed Version:** This column specifies the currently installed version of the software package or library that is affected by the vulnerability.
5. **Fixed Version:** The "Fixed Version" column provides information on the recommended or available fixed version of the software package or library. This assists in determining the necessary updates or patches to address the vulnerabilities.
6. **Links:** Links to relevant external resources or documentation may be included in this column. These links can provide additional context, details, or mitigation guidance for each identified vulnerability.

The structured presentation of this information in HTML format ensures clarity and accessibility, enabling stakeholders to efficiently assess, prioritize, and address security vulnerabilities within the containerized environment. This proactive approach aligns with best practices in container security, ultimately contributing to the overall robustness of containerized applications.

The scanning of the image was completed within a timeframe of 50 seconds. It's important to note, however, that the scanning duration can vary significantly



depending on the size of the image under evaluation. Additionally, the resulting report unveiled a substantial number of vulnerabilities, which naturally stem from the vulnerability database embedded within Trivy. For this reason, users are advised to regularly update the Trivy database to ensure the accuracy of the results.

The integration and configuration of Trivy within the CI/CD pipeline were executed with ease and expediency. This streamlined process underscores the tool's user-friendliness and its seamless compatibility with existing development and deployment workflows.

The variable scanning times, contingent upon image size, highlight the tool's adaptability to different container environments. Users can benefit from Trivy's rapid scanning capabilities for smaller images while appreciating its thoroughness for larger, more complex containers. Regular database updates serve as a proactive measure to uphold the precision of security assessments.

Furthermore, the effortless integration and configuration of Trivy emphasize its value in facilitating efficient DevSecOps practices. This ease of use enables organizations to swiftly incorporate robust container security scanning into their automated CI/CD pipelines, fortifying their overall security posture.

This table summarizes both the positive aspects and considerations when using Trivy for the container analysis.

Aspect	Positive	Negative
Functionality	Effectively scans container images and filesystems for vulnerabilities.	Limited to scanning vulnerabilities in known software packages and libraries.
Vulnerability Database	Maintains an extensive and frequently updated vulnerability database.	Reliance on a centralized database, which may not cover all possible vulnerabilities.
Container Support	Supports various container image formats, including Docker and OCI images.	Focuses primarily on containerized applications, potentially excluding other assets.
Severity Assessment	Assigns severity levels to vulnerabilities, aiding in prioritization.	Severity assessment is based on available data and may not reflect real-world risk.
Customization	Allows for custom policies to tailor scanning to specific requirements.	Customization may require in-depth knowledge and could lead to false negatives.
Integration	Seamlessly integrates with CI/CD pipelines, promoting automation and DevSecOps practices.	Requires effort to configure and integrate into existing workflows.

Aspect	Positive	Negative
Open Source	Being open-source, Trivy is freely available and transparent for code inspection and modification.	Community-supported software may have varying levels of support and documentation.
Report Generation	Generates detailed HTML reports summarizing vulnerabilities for easy analysis.	The quality of the report may depend on the accuracy of the vulnerability database.
Resource Usage	Generally lightweight and doesn't impose a significant resource burden during scans.	Resource usage may vary depending on the size and complexity of container images.
Continuous Updates	Frequent updates to the vulnerability database ensure coverage of the latest threats.	Users must actively update the Trivy database to maintain its effectiveness.
Customization	Allows for custom policies to tailor scanning to specific requirements.	Customization may require in-depth knowledge and could lead to false negatives.
Integration	Seamlessly integrates with CI/CD pipelines, promoting automation and DevSecOps practices.	Requires effort to configure and integrate into existing workflows.

*Table 4: Trivy tool positives and negatives*

## Container Security using Grype

Grype is an open-source tool primarily developed by Aqua Security for scanning vulnerabilities. It targets security issues in container images and filesystems. The tool was conceived in response to the growing popularity of containerization technologies, such as Docker and Kubernetes, and the subsequent need for specialized tools to evaluate the security of containerized applications. Grype is a part of Aqua Security's suite of container security tools [29].

Being an open-source tool, Grype's source code is freely accessible for anyone to use, modify, and contribute to. This open-source nature encourages community participation and allows users to tailor the tool to fit their specific needs [github.com](https://github.com).

When Grype performs a scan, it uses a vulnerability database stored on the local filesystem. This database is constructed by pulling data from a variety of publicly available vulnerability data sources. By default, Grype manages this database and checks for new updates to ensure that every scan uses the most recent vulnerability information. However, this behavior can be configured [30].

Grype requires updated vulnerability information to provide accurate matches. If the local database has not been built within the last 5 days, it will fail execution by default. However, this staleness check can be configured or disabled [30].

In offline or air-gapped environments, Grype can be configured not to perform a network call over the Internet to check for a new database on each run. As long as the vulnerability database and metadata files are placed in the cache directory for the expected schema version, Grype can operate without network access [30].

Grype also provides shell completion through its CLI implementation. The completion code for a shell can be generated by running one of the commands provided [30].

#### Features and Capabilities:

Grype, the open-source vulnerability scanner for container images and filesystems developed by Aqua Security, offers a comprehensive set of features and capabilities to bolster container security. It boasts compatibility with a variety of container image formats, including Docker and OCI images, making it adaptable to diverse containerization technologies and platforms. Grype seamlessly integrates into CI/CD pipelines, container registries, and scanning tools, enabling automated vulnerability assessment throughout the development and deployment lifecycle. Notably, it supports offline scanning, facilitating assessments in environments with limited internet connectivity. Organizations benefit from Grype's flexibility to define custom scanning policies and rules, aligning security practices with specific compliance requirements. Furthermore, Grype produces detailed reports for identified vulnerabilities, equipping users with vital information, including CVE details, severity levels, and recommended remediation steps, to prioritize and rectify security issues effectively.

#### Vulnerability Database:

As it is already mentioned, Grype, the open-source vulnerability scanner for container images and filesystems, relies on external vulnerability databases like the National Vulnerability Database (NVD), Linux distribution databases, open-source vulnerability databases, and commercial feeds to identify known security vulnerabilities. It doesn't maintain its own database but rather leverages these established sources for accurate and up-to-date vulnerability information. Grype's strength lies in its ability to cross-reference these databases, simplifying the process of pinpointing and addressing security risks in containerized environments, benefiting users by ensuring container security through access to comprehensive vulnerability data.

In this particular thesis project, Grype was employed via a pipeline to perform security scanning on the OWASP WebGoat container. The primary objective of this scanning process is to identify vulnerabilities present within the container and, if possible, to propose solutions for mitigating these vulnerabilities.

Configuring Grype was relatively straightforward for this purpose. The only requirements were the installation of Grype on the appserver2 and the creation of a

dedicated pipeline. This pipeline is responsible for invoking Grype, updating its vulnerability database, and conducting a thorough scan of the WebGoat container.

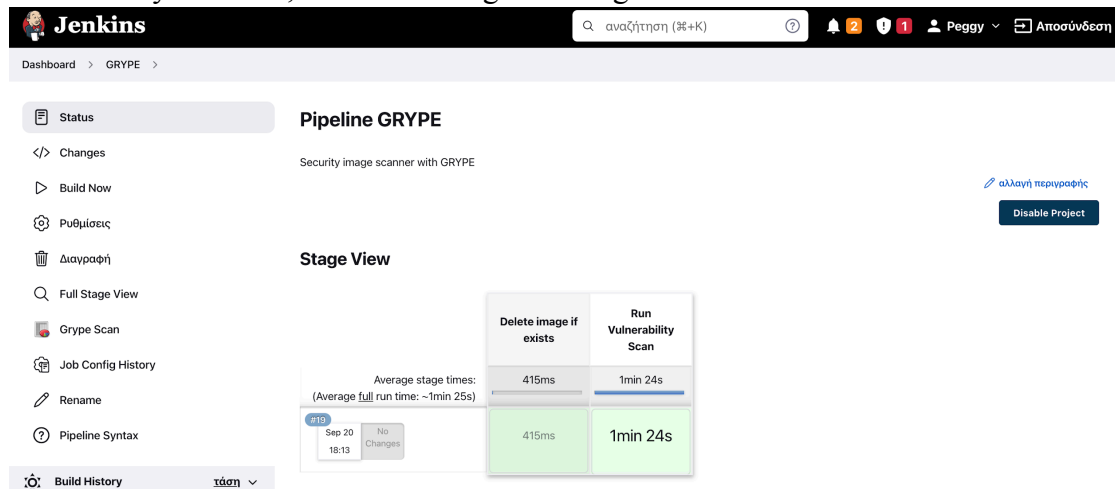


Figure 32: SCA Grype Jenkins Pipeline

The pipeline has been internally configured within the Jenkins environment through the utilization of the following Groovy script:

```

Definition
Pipeline script

Script ?
1 pipeline {
2   agent {
3     label 'ubuntu2'
4   }
5   stages {
6     stage('Delete image if exists') {
7       steps {
8         sh 'echo "Deletion stage"'
9       }
10    }
11    stage('Run Vulnerability Scan') {
12      steps {
13        sh 'mkdir -p reports'
14        sh 'cp /home/appserver2/jenkins-workspace/workspace/htmlGrype.tmp .'
15        sh 'grype webgoat/goatandwolf:latest -o template -t htmlGrype.tmp > reports/grype.html'
16      }
17      publishHTML target : [
18        allowMissing: true,
19        alwaysLinkToLastBuild: true,
20        keepAll: true,
21        reportDir: 'reports',
22        reportFiles: 'grype.html',
23        reportName: 'Grype Scan',
24        reportTitles: 'Grype Scan'
25      ]
26    }
27  }
28 }

```

Figure 33: SCA Grype Jenkins Pipeline configuration

As depicted in the image, this pipeline script consists of stages that perform various tasks, including deleting an image if it exists and running a vulnerability scan using Grype on a Docker image.

Let's break down the script:

- **Agent:** The pipeline is configured to run on an agent labeled as 'ubuntu2.' Agents are nodes or machines where Jenkins runs jobs or tasks.

- Stages:
  - Delete image if exists: This stage is responsible for some form of deletion, but the actual command for deletion is not present in the provided script. It currently echoes "Deletion stage," which is more of a placeholder comment.
  - Run Vulnerability Scan: In this stage, several steps are executed:
    - It creates a directory named 'reports' if it doesn't already exist using the `mkdir -p` command.
    - Copies a template file named 'htmlGrype.tpl' from a specific location to the current workspace.
    - Runs a Grype vulnerability scan on the Docker image 'webgoat/goatandwolf:latest' and saves the scan result in HTML format using the specified template. The results are stored in the 'reports/grype.html' file.
    - Publishes the HTML report generated by Grype using the Jenkins HTML Publisher plugin. This step makes the Grype scan report available in the Jenkins job's build artifacts.

As previously mentioned, the report detailing the identified vulnerabilities is made available on the Jenkins platform and is denoted as "Grype Scan."

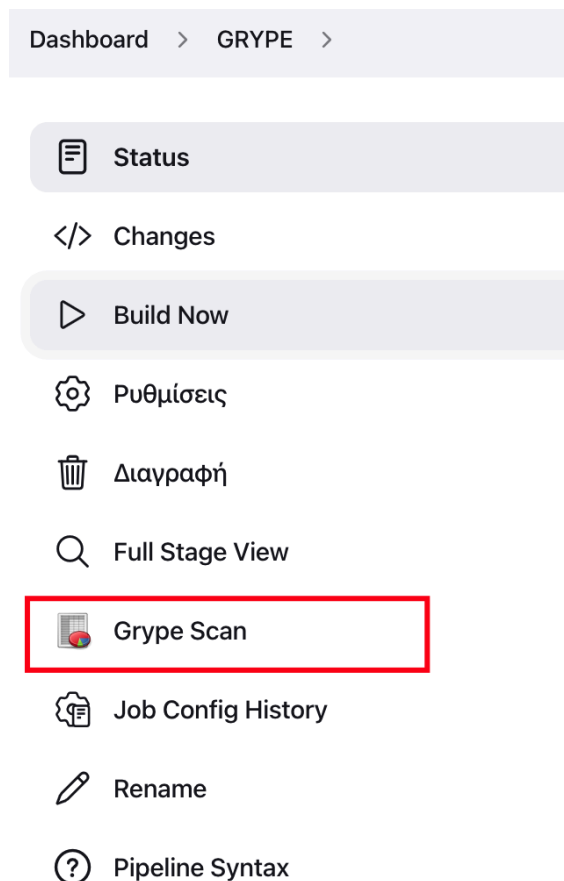


Figure 34: SCA Grype Report in Jenkins Pipeline Menu

**Identified Vulnerabilities**

Search in all Fields.....

NAME	INSTALLED	TYPE	VULNERABILITY	SEVERITY	DESCRIPTION	STATE	FIXED IN
"tar"	"1.34+dfsg-1"	"deb"	"CVE-2005-2541"	"Negligible"	"Tar 1.15.1 does not properly warn the user when extracting setuid or setgid files, which may allow local users or remote attackers to gain privileges."	"not-fixed"	"[]"
"jquery"	"3.5.1"	"java-archive"	"CVE-2007-2379"	"Medium"	"The jQuery framework exchanges data using JavaScript Object Notation (JSON) without an associated protection scheme, which allows remote attackers to obtain the data via a web page that retrieves the data through a URL in the SRC attribute of a SCRIPT element and captures the data using other JavaScript code, aka "JavaScript Hijacking."	"unknown"	"[]"
"login"	"1:4.8.1-1"	"deb"	"CVE-2007-5686"	"Negligible"	"initscripts in rPath Linux 1 sets insecure permissions for the /var/log/btmp file, which allows local users to obtain sensitive information regarding authentication attempts. NOTE: because sshd detects the insecure permissions and does not log certain events, this also prevents sshd from logging failed authentication attempts by remote attackers."	"not-fixed"	"[]"
"passwd"	"1:4.8.1-1"	"deb"	"CVE-2007-5686"	"Negligible"	"initscripts in rPath Linux 1 sets insecure permissions for the /var/log/btmp file, which allows local users to obtain sensitive information regarding authentication attempts. NOTE: because sshd detects the insecure permissions and does not log certain events, this also prevents sshd from logging failed authentication attempts by remote attackers."	"not-fixed"	"[]"
"libssl1.1"	"1.1.1k-1+deb11u1"	"deb"	"CVE-2007-6755"	"Negligible"	"The NIST SP 800-90A default statement of the Dual Elliptic Curve Deterministic Random Bit Generation (Dual_EC_DRBG) algorithm contains point Q constants with a possible relationship to certain "skeleton key" values, which might allow context-dependent attackers to defeat cryptographic protection mechanisms by leveraging knowledge of those values. NOTE: this is a preliminary CVE for Dual_EC_DRBG; future research may provide additional details about point Q and associated attacks, and could potentially lead to a RECAST or REJECT of this CVE."	"not-fixed"	"[]"
"openssl"	"1.1.1k-1+deb11u1"	"deb"	"CVE-2007-6755"	"Negligible"	"The NIST SP 800-90A default statement of the Dual Elliptic Curve Deterministic Random Bit Generation (Dual_EC_DRBG) algorithm contains point Q constants with a possible relationship to certain "skeleton key" values, which might allow context-dependent attackers to defeat cryptographic protection mechanisms by leveraging knowledge of those values. NOTE: this is a preliminary CVE for Dual_EC_DRBG; future research may provide additional details about point Q and associated attacks, and could potentially lead to a RECAST or REJECT of this CVE."	"not-fixed"	"[]"
"libnginx-mod-http-geoip"	"1.18.0-6.1"	"deb"	"CVE-2009-4487"	"Negligible"	"nginx 0.7.64 writes data to a log file without sanitizing non-printable characters, which might allow remote attackers to modify a window's title, or possibly execute arbitrary commands or overwrite files, via an HTTP request containing an escape sequence for a terminal emulator."	"not-fixed"	"[]"

*Figure 35: SCA Grype Report*

The "Grype Scan" report is produced in HTML format and serves as a vital resource for conducting a thorough vulnerability assessment. This comprehensive report consists of several columns, each containing crucial information for evaluating the security of containerized software components. These columns are as follows:

1. **NAME:** This column displays the name of the software component being assessed. It helps identify the specific component that may be susceptible to vulnerabilities.
2. **INSTALLED:** The "INSTALLED" column provides details about the installed version of the software component within the container image. Understanding the installed version is essential for determining whether updates or patches are required.
3. **TYPE:** In the "TYPE" column, you'll find information regarding the classification or type of the software component. This categorization aids in assessing the significance of the vulnerability.
4. **VULNERABILITY:** This column lists the name of the identified vulnerability associated with the software component. Knowing the specific vulnerability is crucial for assessing its potential impact and mitigation steps.
5. **SEVERITY:** The "SEVERITY" column assigns a severity level to the vulnerability. Severity levels can range from low to critical and help prioritize remediation efforts based on the potential risk.
6. **DESCRIPTION:** Within the "DESCRIPTION" column, you'll find a concise description of the identified vulnerability. This description offers insights into the nature of the security issue.
7. **STATE:** The "STATE" column indicates the current status or state of the vulnerability. It provides information about whether the vulnerability is active or has been addressed.
8. **FIXED IN:** The "FIXED IN" column specifies the version in which the vulnerability has been resolved or fixed. This information is critical for

determining the appropriate action to take, such as updating to a secure version.

By examining these columns in the "Grype Scan" report, security professionals and administrators can gain a comprehensive understanding of the security posture of containerized software components and take informed steps to mitigate potential risks.

The scanning of the image is completed within an average duration of 5 minutes. It's noteworthy to mention that the scanning duration is directly proportional to the size of the image being scanned. In other words, the scanning process aligns with the image's dimensions, ensuring an efficient and proportional assessment.

It's worth noting that Grype doesn't require an internet connection to operate. Once it has been installed on the host system, it can run independently without the need for network connectivity. This offline capability enhances its flexibility, allowing it to perform vulnerability scanning even in isolated or air-gapped environments where internet access may be restricted or unavailable.

As previously mentioned, the scanning for vulnerabilities is conducted based on the Grype vulnerability database. It's important to clarify that this database is generated by external sources and is not an integral part of the tool itself. Therefore, to ensure reliable results, it is imperative for the user to regularly update this database. By keeping the vulnerability database up-to-date, users can maintain the accuracy and effectiveness of the scanning process, enhancing the overall security assessment of the containerized environment.

The integration of Grype into the pipeline was a relatively straightforward process. It involved creating a dedicated mini-pipeline within Jenkins, through which the Grype tool is invoked, and seamlessly linking it with the larger OWASP WebGoat pipeline. Furthermore, the resulting report is quite legible and user-friendly. One valuable addition, currently absent but worth considering, would be information on how to resolve or mitigate the identified vulnerabilities. Including guidance on addressing vulnerabilities would enhance the utility of the report, providing a comprehensive solution for security concerns.

The table below provides a concise overview of Grype's strengths and potential limitations, helping users assess its suitability for container security needs and environments.

Aspect	Positive Aspects	Negative Aspects
Functionality	Effectively scans container images and filesystems for vulnerabilities.	Limited to scanning vulnerabilities in known software packages and libraries.
Vulnerability Database	Maintains an extensive and frequently updated vulnerability database.	Reliance on a centralized database, which may not cover all possible vulnerabilities.

Aspect	Positive Aspects	Negative Aspects
Container Support	Supports various container image formats, including Docker and OCI images.	Focuses primarily on containerized applications, potentially excluding other assets.
Severity Assessment	Assigns severity levels to vulnerabilities, aiding in prioritization.	Severity assessment is based on available data and may not reflect real-world risk.
Customization	Allows for custom policies to tailor scanning to specific requirements.	Customization may require in-depth knowledge and could lead to false negatives.
Integration	Seamlessly integrates with CI/CD pipelines, promoting automation and DevSecOps practices.	Requires effort to configure and integrate into existing workflows.
Open Source	Being open-source, gRYPE is freely available and transparent for code inspection and modification.	Community-supported software may have varying levels of support and documentation.
Report Generation	Generates detailed reports summarizing vulnerabilities for easy analysis.	The quality of the report may depend on the accuracy of the vulnerability database.
Resource Usage	Generally lightweight and doesn't impose a significant resource burden during scans.	Resource usage may vary depending on the size and complexity of container images.
Continuous Updates	Frequent updates to the vulnerability database ensure coverage of the latest threats.	Users must actively update the gRYPE database to maintain its effectiveness.

*Table 5: Grype tool positives and negatives*



## DAST Security using OWASP-Zap

The Open Web Application Security Project (OWASP) Zed Attack Proxy (ZAP) is a no-cost, open-source DAST tool made specifically for scanning websites for vulnerabilities. In the OWASP community, it is one of the most active projects and is managed by The Software Security Project (SSP).

ZAP can be classified as a "man-in-the-middle proxy." The intermediary entity is situated in the intermediary position, positioned between the browser utilized by the tester and the web application being tested. Its primary function is to intercept and scrutinize the communications that are sent between these two entities. The system has the capability to make necessary modifications to these messages prior to transmitting them to the intended recipient. The utilization of proxy capability proves to be advantageous in discerning the behavioral patterns of an application when subjected to altered inputs, hence aiding in the detection of plausible security flaws. [31]

The ZAP tool offers a wide range of capabilities that cater to individuals with varying degrees of expertise, including those who are new to security testing as well as seasoned security testing professionals. The software provides compatibility with several prominent operating systems and Docker, hence offering flexibility in terms of operating system selection. A diverse range of supplementary features may be accessed through various add-ons in the ZAP Marketplace, which can be conveniently accessed from inside the ZAP client. [31]

The key elements of ZAP encompass:

- The ZAP tool offers spidering functionality, which involves the use of spiders to systematically explore online applications. These spiders analyze the HTML content of web application answers in order to identify and uncover linkages inside the application. In the context of AJAX applications, it is probable that ZAP's AJAX spider would exhibit more efficacy due to its ability to investigate the web application by activating browsers, which then go via the created links. [31]
- The ZAP tool does passive scanning by analyzing all requests and answers that are proxied through it, with the purpose of detecting any possible vulnerabilities. Additionally, the system provides support for active scanning, a technique that aims to identify vulnerabilities by employing established attack methods on the designated targets. Nevertheless, it is imperative to restrict the utilization of active scanning only to authorized targets, since it has the potential to jeopardize the security of these targets. [31]
- The Heads Up Display (HUD) is a novel user interface that enables direct access to ZAP functionality within a web browser. Zaproxy.org is a highly beneficial tool for those who are new to online security, as well as for experienced penetration testers. It enables the latter group to concentrate on assessing an application's performance while simultaneously offering crucial security information and functionality. [31]
- Automation: The utilization of ZAP as a tool in the context of automation is highly advantageous, as it offers support for a diverse array of alternatives. [31]

The present thesis project used the OWASP tool "ZAP" to do dynamic analysis on the OWASP WebGoat application. A distinct pipeline was established, referred to as the auxiliary pipeline of OWASP WebGoat, and executed on the Ubuntu2 agent.

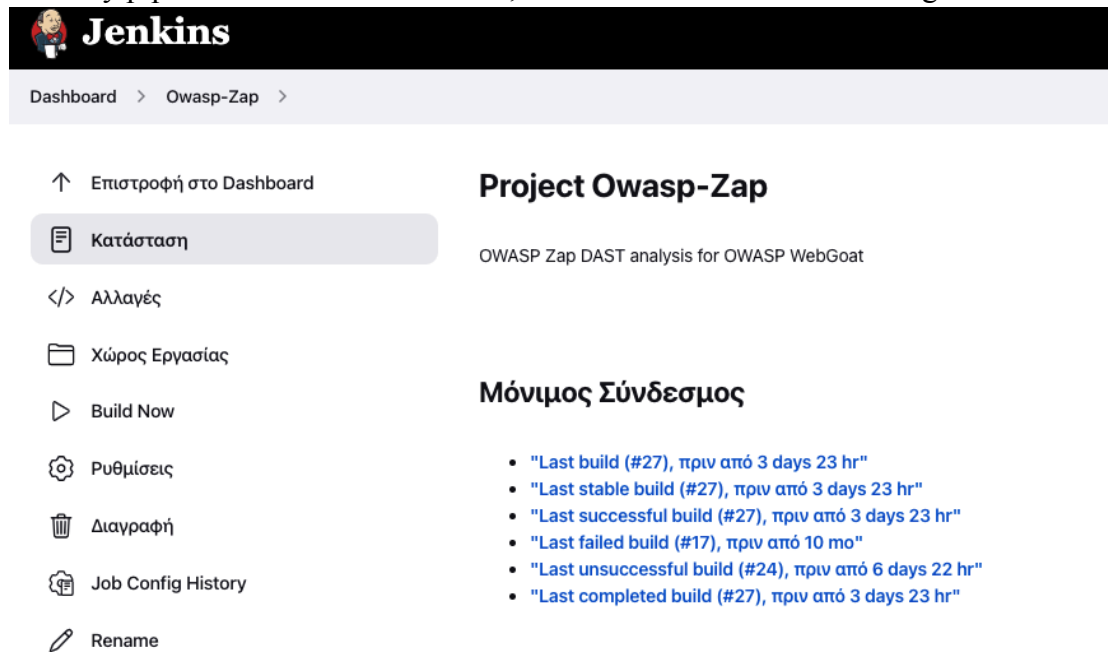


Figure 36: DAST OWASP-Zap Jenkins Pipeline

The pipeline has been setup to execute a Bash script for the purpose of doing dynamic analysis on the application.



Figure 37: DAST OWASP-Zap execution bash script

The provided script is written in the Bash programming language. The command executed is "docker run -v \$(pwd):/zap/wrk/:rw -t owasp/zap2docker-stable zap-baseline.py -t". Please provide the URL "http://192.168.2.15:8080/WebGoat/" to generate an HTML report.

In order to analyze the given instructions, let us proceed with a detailed examination:

- The command "docker run" is utilized to initiate the execution of a Docker container.

- The specified portion of the command facilitates the mounting of the present working directory, obtained through the \$(pwd) command, onto the /zap/wrk/ directory within the Docker container. This enables the seamless transfer of files between the local system and the container. This functionality enables the container to do read and write operations on files within the local directory. The option "rw" signifies that the volume is mounted with read-write capabilities.
- The use of the "-t" option results in the allocation of a pseudo-TTY terminal within the container.
- The OWASP/ZAP2DOCKER-STABLE repository is being referred to. The Docker image now being executed is identified by the name and encompasses the OWASP ZAP framework.
- The script being executed is zap-baseline.py, which pertains to ZAP. The provided script serves as a fundamental tool for doing security testing.
- The user provided a URL: "http://192.168.2.15:8080/WebGoat/". The provided information indicates the desired Uniform Resource Locator (URL) of the online application that is intended to be subjected to scanning. In this particular scenario, the task involves doing a scan on a web application that is being hosted at the following URL: http://192.168.2.15:8080/WebGoat/.
- The "report\_html" option in ZAP allows the user to choose the format and location of the produced report. In this particular instance, an HTML report is being generated and assigned the filename "report\_html".

Upon execution of the command, the OWASP ZAP Docker container will launch and proceed to do a baseline security scan on the designated web application, which can be accessed through the URL http://192.168.2.15:8080/WebGoat/. Subsequently, an HTML report titled "report\_html" will be generated. The report will encompass details on security vulnerabilities that were identified during the scanning process.

In order to perform this command, it is necessary to ensure that two conditions are met: 1) the application is actively running, and 2) the Docker container containing the ZAP tool is installed on the computer.

## Workspace of Owasp-Zap on ubuntu2

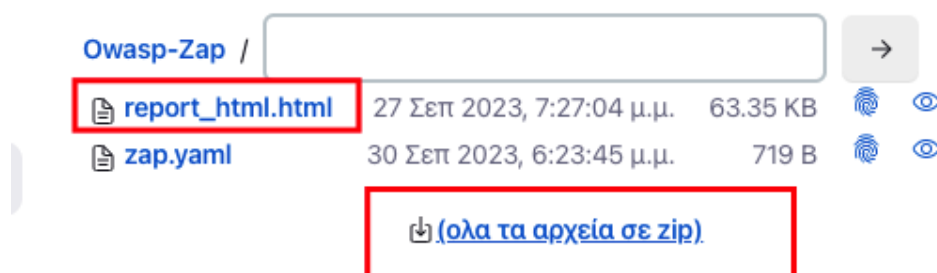


Figure 38: DAST OWASP-Zap report download

The ZAP report is created within the workspace of appserver2, where the execution of the ZAP report takes place. In order to access the report, individuals have the option to either choose the single file or alternatively opt to download the complete zip folder. When the report is accessed using Jenkins, it is observed that the content is shown in plain text format, lacking any visual formatting. This is attributed to the report being an independent file that does not include the associated CSS files necessary for style.

### ZAP Scanning Report

Site: <http://192.168.2.15:8080>

Generated on Wed, 27 Sep 2023 19:27:03

ZAP Version: 2.13.0

#### Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	3
Informational	5
False Positives:	0

#### Alerts

Name	Risk Level	Number of Instances
<a href="#">Absence of Anti-CSRF Tokens</a>	Medium	5
<a href="#">Content Security Policy (CSP) Header Not Set</a>	Medium	5
<a href="#">Cookie No HttpOnly Flag</a>	Low	1
<a href="#">Cookie without SameSite Attribute</a>	Low	1
<a href="#">Permissions Policy Header Not Set</a>	Low	5
<a href="#">Authentication Request Identified</a>	Informational	1
<a href="#">Non-Storable Content</a>	Informational	3
<a href="#">Session Management Response Identified</a>	Informational	2
<a href="#">Storable and Cacheable Content</a>	Informational	8
<a href="#">User Controllable HTML Element Attribute (Potential XSS)</a>	Informational	5

#### Alert Detail

##### Medium

No Anti-CSRF tokens were found in a HTML submission form.

##### Absence of Anti-CSRF Tokens

A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSSRF, one-click attack, session riding, confused deputy, and sea surf.

CSRF attacks are effective in a number of situations, including:

Figure 39: DAST OWASP-Zap scanning report

Conversely, when the report is accessed using the user's file explorer subsequent to downloading the data as a compressed archive to their own computer, it exhibits enhanced visual appeal, rendering it more engaging and facilitating effortless comprehension.

# ZAP Scanning Report

Site: <http://192.168.2.15:8080>

Generated on Wed, 27 Sep 2023 19:27:03

ZAP Version: 2.13.0

## Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	3
Informational	5
False Positives:	0

## Alerts

Name	Risk Level	Number of Instances
Absence of Anti-CSRF Tokens	Medium	5
Content Security Policy (CSP) Header Not Set	Medium	5
Cookie No HttpOnly Flag	Low	1
Cookie without SameSite Attribute	Low	1
Permissions Policy Header Not Set	Low	5
Authentication Request Identified	Informational	1
Non-Storable Content	Informational	3
Session Management Response Identified	Informational	2
Storable and Cacheable Content	Informational	8
User Controllable HTML Element Attribute (Potential XSS)	Informational	5

Figure 40: OWASP-Zap scanning report results (Summary)

Alert Detail	
Medium	<b>Absence of Anti-CSRF Tokens</b>
Description	<p>No Anti-CSRF tokens were found in a HTML submission form.</p> <p>A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.</p> <p>CSRF attacks are effective in a number of situations, including:</p> <ul style="list-style-type: none"> <li>* The victim has an active session on the target site.</li> <li>* The victim is authenticated via HTTP auth on the target site.</li> <li>* The victim is on the same local network as the target site.</li> </ul> <p>CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy.</p>
URL	<a href="http://192.168.2.15:8080/WebGoat/">http://192.168.2.15:8080/WebGoat/</a>
Method	GET
Parameter	
Attack	
Evidence	<form action="/WebGoat/login" method="POST" style="width: 200px;">
Other Info	No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "exampleinputEmail1" "exampleInputPassword1"]
URL	<a href="http://192.168.2.15:8080/WebGoat/login">http://192.168.2.15:8080/WebGoat/login</a>
Method	GET

Figure 41: DAST OWASP-Zap scanning report results (Alert details)

The report has introductory details, including general information, the title of the scanned page, the date of the scan, and the version of the zap tool employed. Subsequently, it demonstrates:

1. Briefly, in a table, the number of alerts found according to their risk level.
2. A table with the alerts that were found, their level, and how many times they appeared in the application.
3. Details for each alert. The details for each alert include:
  1. The severity of the Alert and its name.
  2. Description: A description of the alert.

3. URL: The Uniform Resource Locator (URL) that the tool accessed and the specific location where this warning was detected. The URL section includes:
  1. Method: Information on the method employed, such as GET, POST, or DELETE.
  2. Parameter: The parameters that were received by the URL.
  3. Attack: The term "attack" encompasses a range of malevolent behaviors or security weaknesses.
  4. Evidence: The "Evidence" section presents the following HTML code as substantiation for a certain configuration or element found within a web application or webpage. Within the realm of web security assessment or analysis, this particular evidence has significant importance in the identification of possible vulnerabilities, comprehension of web page structure, and determination of data transmission mechanisms between the client and server.
  5. Other info: Additional details on the URL.
4. Instances: Instances refer to the frequency with which a particular warning has occurred. It pertains to the number of times this alert has been observed or recorded.
5. Solution: The alert is accompanied with a proposed solution.
6. Reference: A hyperlink that provides the user with valuable information pertaining to the alert.
7. CWE id: The CWE id is a distinct identification that is allocated to a particular software or cybersecurity flaw or vulnerability within the framework of the Common flaw Enumeration (CWE) protocol.
8. WASC id: Each WASC id corresponds to a distinct category of online application security issue, such as cross-site scripting (XSS), SQL injection, or cross-site request forgery (CSRF). These identifiers are often employed by security practitioners, scholars, and institutions to denote and analyze web application security risks in a consistent fashion.
9. Plugin id: The term "Plugin id" denotes the unique identification linked to a particular security plugin or rule.

Additionally, the alerts are arranged in a descending sequence based on the level of risk.

The process of integrating the ZAP tool with Jenkins was uncomplicated, achieved by utilizing a script that directed to the pre-existing installation of Docker ZAP on the Ubuntu2 computer. In order to upgrade the ZAP tool, it is necessary for the user to uninstall the existing Docker instance and thereafter obtain a different version.

The duration of the dynamic analysis conducted with the ZAP tool is estimated to be around 10 minutes.

The report exhibits a high level of readability when being downloaded into the user's PC. Accessing the report provided by Jenkins is feasible, although it may not be as enjoyable.

Here's a table summarizing various aspects, both positive and negative, of using the ZAP tool for dynamic analysis:

Aspect	Positive Aspects	Negative Aspects
Vulnerability Detection	Effectively identifies web application vulnerabilities	Potential for false positives that require manual review
Ease of Integration	Straightforward integration with Jenkins and Docker	Upgrading the ZAP tool may require Docker cleanup
Analysis Duration	Dynamic analysis completes relatively quickly (around 10 mins)	Analysis time may vary based on application complexity
Reporting	Structured and detailed reports with plugin IDs	Jenkins-based report viewing may be less user-friendly
Security Assessment	Provides insights into the security posture of web apps	Relies on existing knowledge of web application security
Customization	Configurable for specific testing and scanning parameters	Requires expertise for fine-tuning and optimal results
Compatibility	Supports a wide range of web technologies and frameworks	May require updates for compatibility with new tech
Scripting and Automation	Offers scripting capabilities for automation and CI/CD	Scripting complexity may vary depending on use case
Resource Usage	Moderately resource-efficient during scanning	Resource utilization may increase with complex apps
Open Source	ZAP is open-source, fostering a supportive community	Limited official support, community-dependent

*Table 6:OWASP-Zap tool positives and negatives*

## DAST Security using Arachni

Arachni is an online application security scanner framework that is open-source and high-performance. Its primary purpose is to aid in the evaluation of web application security. The primary purpose of its creation was to serve as an educational exercise and a tool for conducting targeted security tests on online applications, with the aim of identifying, categorizing, and documenting security vulnerabilities. Over the course of its development, the infrastructure has seen significant advancements, resulting in a robust system that can effectively conduct security audits for web applications and do common data scraping tasks with a high level of reliability. [32]

Arachni is frequently employed as a tool for Dynamic Application Security Testing (DAST). The Dynamic Application Security Testing (DAST) method is a form of black-box application testing that enables the testing of applications in real-time operational conditions. In the context of application testing using Dynamic Application Security Testing (DAST), it is not necessary to possess access to the source code in order to identify vulnerabilities. The activity being referred to is often known as a penetration test, which involves the identification of vulnerabilities and misconfigurations in an application's external environment, as seen by a potential attacker. [33]

The capabilities of Arachni encompass:

- Rapid identification of vulnerabilities in web applications.
- The system provides support for a range of report export formats, including PDF, text, JSON, XML, YAML, and others.
- The system is designed to accommodate a variety of technologies, such as HTML5, DOM manipulation, AJAX, and JavaScript.
- Incorporating the flexibility to modify the limits of page count, DOM depth, directory depth, and redirect inside the system.
- The process of auditing include the examination of many elements such as links, forms, user-interface inputs, cookies, headers, JSON request data, link templates, and other relevant components.
- The process of incorporating and providing assistance for intricate web applications. [34]

Arachni has a high degree of scalability and modularity, enabling its utilization as a rudimentary command-line scanning tool or its configuration as a robust scanning grid for conducting extensive application security testing procedures on a wide scale. Additionally, it possesses an intelligent, self-adaptive capacity. The tool enhances its performance through a process of self-training by analyzing and assimilating information from HTTP replies. This iterative learning approach leads to improved accuracy in evaluations and reduces the occurrence of false-positive outcomes. [35]

Nevertheless, it is important to acknowledge that the tool's most recent update occurred in 2017, and subsequent to that, no official information pertaining to any further updates has been made available. This factor has the potential to impact the efficacy of the system in addressing contemporary vulnerabilities and advancements in web technology. [34]

In summary, Arachni is a robust Dynamic Application Security Testing (DAST) solution that facilitates comprehensive security assessments of online applications. The extensive array of features and capabilities of this program renders it a significant asset in the identification and mitigation of potential security issues. Nevertheless, the absence of latest updates necessitates users to augment it with additional tools or methodologies in order to guarantee thorough security coverage.

The present thesis employed the Arachni tool to conduct dynamic analysis on the OWASP WebGoat application using the Jenkins platform. A supplementary pipeline was developed, which is triggered by the primary OWASP WebGoat pipeline and executes the Arachni tool.



Figure 42: DAST Arachni Jenkins Pipeline

The incorporation of Arachni into Jenkins for the purpose of dynamic application security testing (DAST) within the framework of the OWASP WebGoat application represents a noteworthy advancement in bolstering the security stance of online applications. An automated and scalable strategy has been built by integrating the Arachni tool into the OWASP WebGoat pipeline through the creation of an external pipeline. This integration allows for the easy identification and resolution of security issues.

The process of integrating with Jenkins was quite uncomplicated. The pipeline was set to execute on vm-1, which serves as the application server.

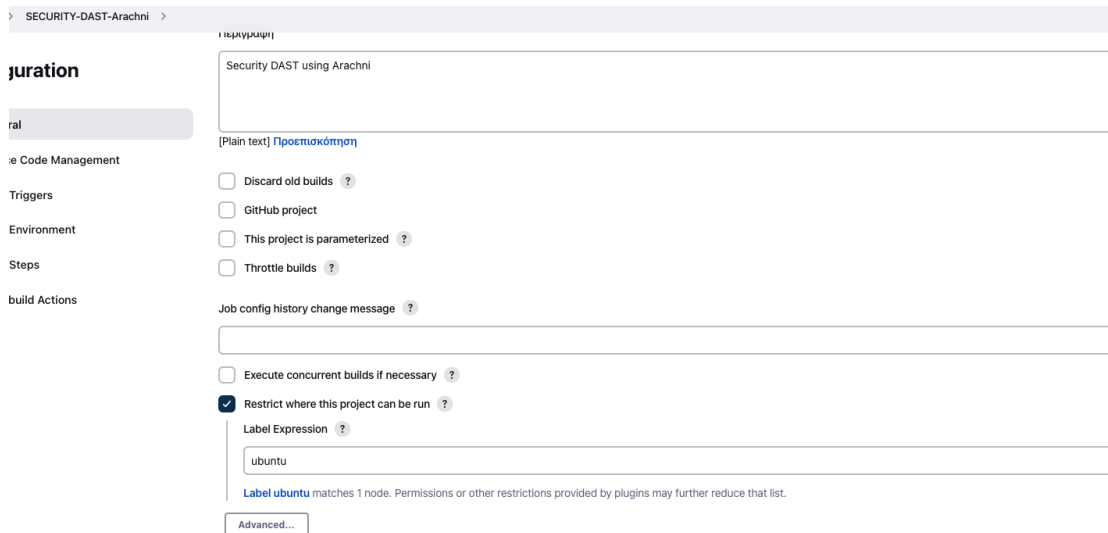


Figure 43: DAST Arachni Jenkins Pipeline configuration

Furthermore, the execution of ARACHNI was facilitated by the utilization of a Bash script. The provided script is written in the Bash programming language.

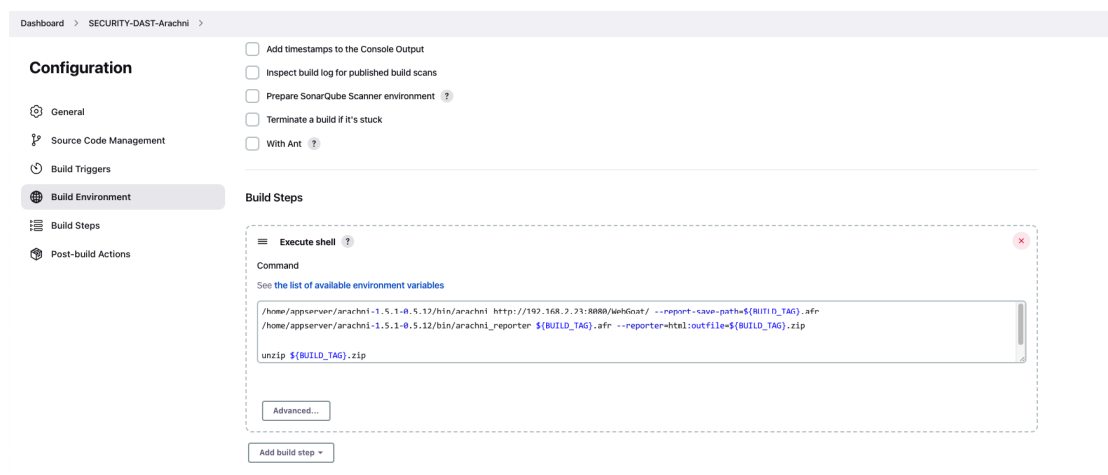


Figure 44: DAST Arachni execution bash script

```

/home/appserver/arachni-1.5.1-0.5.12/bin/arachni
http://192.168.2.23:8080/WebGoat/ --report-save-path=${BUILD_TAG}.afr
/home/appserver/arachni-1.5.1-0.5.12/bin/arachni_reporter ${BUILD_TAG}.afr
--reporter=html:outfile=${BUILD_TAG}.zip
unzip ${BUILD_TAG}.zip

```

The integration process was well managed, and a Bash script was employed to automate the execution of ARACHNI within the Jenkins pipeline. The provided script aims to optimize the procedure of doing a comprehensive scan on the OWASP WebGoat application in order to identify potential security flaws.

The following is an analysis of the script's functionality:

- The initiation of ARACHNI is achieved by invoking the ARACHNI executable, which is situated in the directory path /home/appserver/arachni-1.5.1-0.5.12/bin/arachni.
- The provided information designates the specific URL that is to be scanned, which is http://192.168.2.23:8080/WebGoat/.
- The ARACHNI report is saved with the file name \${BUILD\_TAG}.afr, where \${BUILD\_TAG} is an environment variable in the Jenkins system that serves as a distinct identifier for each individual Jenkins build.
- Once the ARACHNI scan has concluded, the script proceeds to build an HTML report by utilizing the arachni\_reporter command. The input AFR file is specified, which holds the scan results, and the output file is designated as \${BUILD\_TAG}.zip. The present HTML report offers a complete and thorough examination of the security results.

Ultimately, the script decompresses the HTML report that has been prepared, so enabling its accessibility for subsequent study or distribution. The utilization of this automated procedure not only expedites the evaluation of security measures but also guarantees uniformity and the ability to replicate the outcomes.

The pipeline's execution time of 2 hours and 10 minutes highlights the thoroughness and completeness of the security testing procedure. During this period, Arachni conducted a thorough and comprehensive scan of the OWASP WebGoat application, extensively scrutinizing several aspects in search of probable vulnerabilities.

The produced report assumes a key role as an essential artifact upon the successful completion of the pipeline. This report presents the facts and insights obtained after a thorough scan, offering a full picture of the security status of the application. The object in question serves as a concrete manifestation of the meticulousness and accuracy utilized in the identification and evaluation of possible hazards.

The report may be found within the Jenkins workspace, where the tool was ran. Regrettably, the report lacks readability when accessible via the Jenkins user interface, as it lacks essential .css files and fonts necessary for enhancing user-friendliness and visual aesthetics. In contrast, when a person downloads the report as a compressed file and accesses the .html file locally, they are able to observe the report in its entirety, including the full range of colors and images it presents.

http://192.168.2.15:8080/WebGoat/ Generated on 2023-09-27 21:42:20 +0000

### Summary

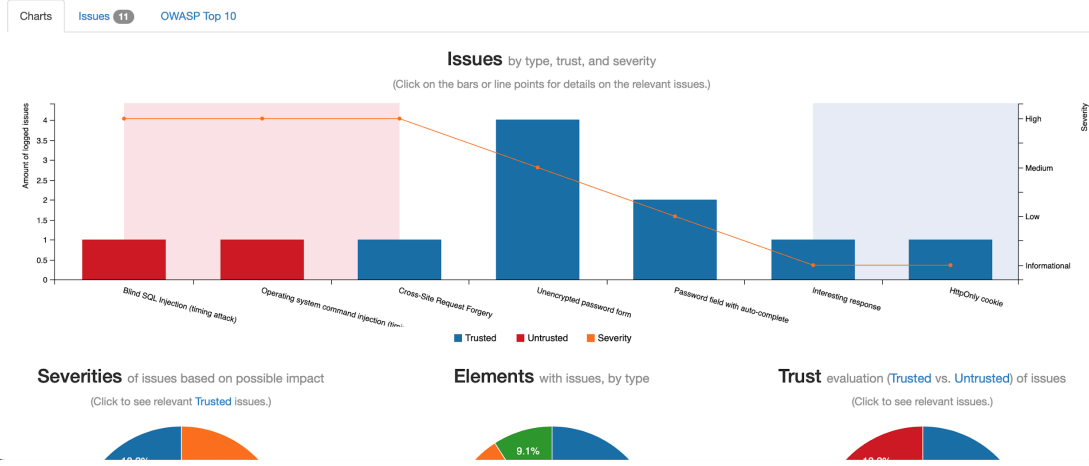


Figure 45: DAST Arachni report summary

The above visual representation illustrates that the Arachni report encompasses five distinct tabs, which afford the user the ability to traverse through and collect pertinent information pertaining to the identified faults and vulnerabilities inside the application. The initial tab presents a concise overview encompassing relevant details such as the application's URL, the date of the scan, and three further sub-tabs.

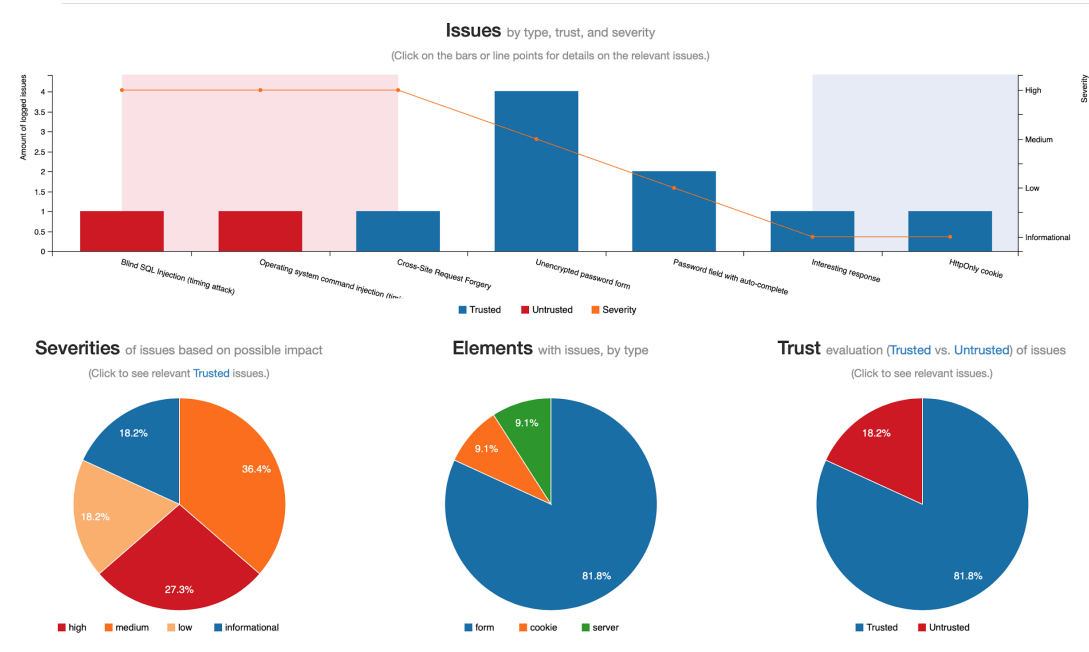


Figure 46: DAST Arachni report issues chart

The charts present a visual representation of various concerns categorized by kind, trustworthiness, and severity. Additionally, they depict the distribution of severities, components, and levels of trust.

The identified concerns have been classified into three categories based on their severity: high, medium, and low.

The purpose of this inquiry is to determine the specific concerns that are included in the OWASP Top Ten list.

The second tab is dedicated to addressing the various issues, with sub-tabs organized based on the level of vulnerability and distinguishing between trusted and untrusted issues.

Arachni v1.5.1 Summary Issues (3) Plugin results (1) Sitemap (10) Configuration (Found a false positive? Please let us know.)

<http://192.168.2.15:8080/WebGoat/> Generated on 2023-09-27 21:42:20 +0000

### Issues

Trusted (3) Untrusted (2)

At the time these issues were logged there were no abnormal interferences or anomalous server behavior. These issues are considered trusted and accurate.

High (1) Medium (4) Low (2) Informational (2)

#### Cross-Site Request Forgery (1) csrf

In the majority of today's web applications, clients are required to submit forms which can perform sensitive operations. An example of such a form being used would be when an administrator wishes to create a new user for the application. In the simplest version of the form, the administrator would fill-in:

- Name
- Password
- Role (level of access)

Continuing with this example, Cross Site Request Forgery (CSRF) would occur when the administrator is tricked into clicking on a link, which if logged into the application, would automatically submit the form without any further interaction.

Cyber-criminals will look for sites where sensitive functions are performed in this manner and then craft malicious requests that will be used against clients via a social engineering attack.

There are 3 things that are required for a CSRF attack to occur:

1. The form must perform some sort of sensitive action.
2. The victim (the administrator the example above) must have an active session.
3. Most importantly, all parameter values must be **known** or **guessable**.

References

- CWE-352
- Wikipedia
- OWASP
- CGI Security

Figure 47: DAST Arachni report issues

The third tab, which is specifically designated for plugin findings, plays a vital role in the Arachni report by offering significant information about the security status of the web application that was scanned. This section provides a comprehensive analysis of the URLs included inside the application, focusing on their safety status and highlighting any security vulnerabilities.

Arachni v1.5.1 Summary Issues (3) Plugin results (1) Sitemap (10) Configuration (Found a false positive? Please let us know.)

<http://192.168.2.15:8080/WebGoat/> Generated on 2023-09-27 21:42:20 +0000

### Plugin results

Health map

Generates a simple list of safe/unsafe URLs.

Total	9	<a href="http://192.168.2.15:8080/WebGoat/">http://192.168.2.15:8080/WebGoat/</a>
Without issues	6	<a href="http://192.168.2.15:8080/WebGoat/login">http://192.168.2.15:8080/WebGoat/login</a>
With issues	3	<a href="http://192.168.2.15:8080/WebGoat/register.mvc">http://192.168.2.15:8080/WebGoat/register.mvc</a>
Issue percentage	33	<a href="http://192.168.2.15:8080/WebGoat/css/animate.css">http://192.168.2.15:8080/WebGoat/css/animate.css</a>
		<a href="http://192.168.2.15:8080/WebGoat/css/font-awesome.min.css">http://192.168.2.15:8080/WebGoat/css/font-awesome.min.css</a>
		<a href="http://192.168.2.15:8080/WebGoat/css/main.css">http://192.168.2.15:8080/WebGoat/css/main.css</a>
		<a href="http://192.168.2.15:8080/WebGoat/plugins/bootstrap/css/bootstrap.min.css">http://192.168.2.15:8080/WebGoat/plugins/bootstrap/css/bootstrap.min.css</a>
		<a href="http://192.168.2.15:8080/WebGoat/registration">http://192.168.2.15:8080/WebGoat/registration</a>
		<a href="http://192.168.2.15:8080/WebGoat/start.mvc">http://192.168.2.15:8080/WebGoat/start.mvc</a>

Figure 48: DAST Arachni report plugin results

The sitemap, which is included in the fourth tab of the Arachni report, plays a crucial role in comprehending the architecture and navigation routes of the online application that was subjected to scanning. The sitemap functions as a graphical depiction of the application's structure, presenting a hierarchical overview of URLs, folders, and their interconnectedness.

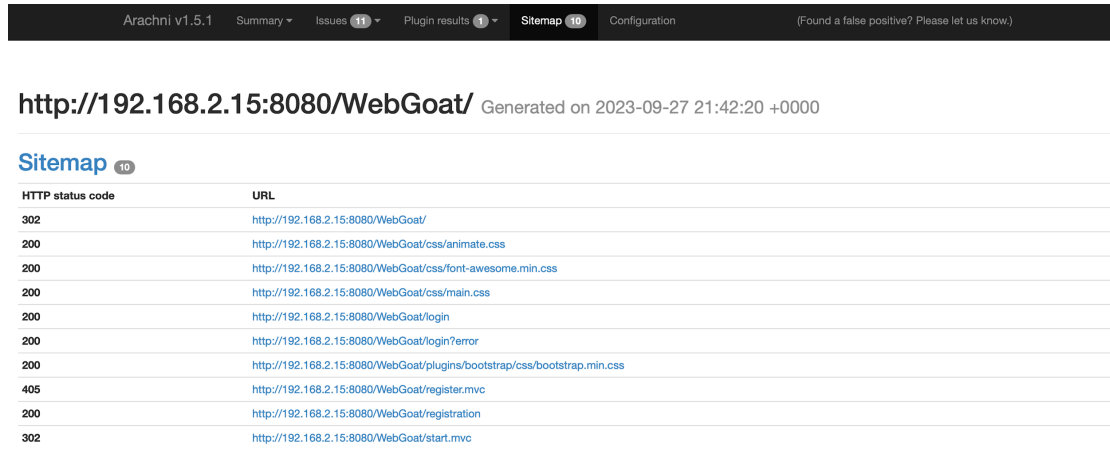


Figure 49: DAST Arachni report sitemap

The last section of the Arachni report is the configuration tab, which offers customers a comprehensive overview of the factors and settings that dictated the scanning procedure.

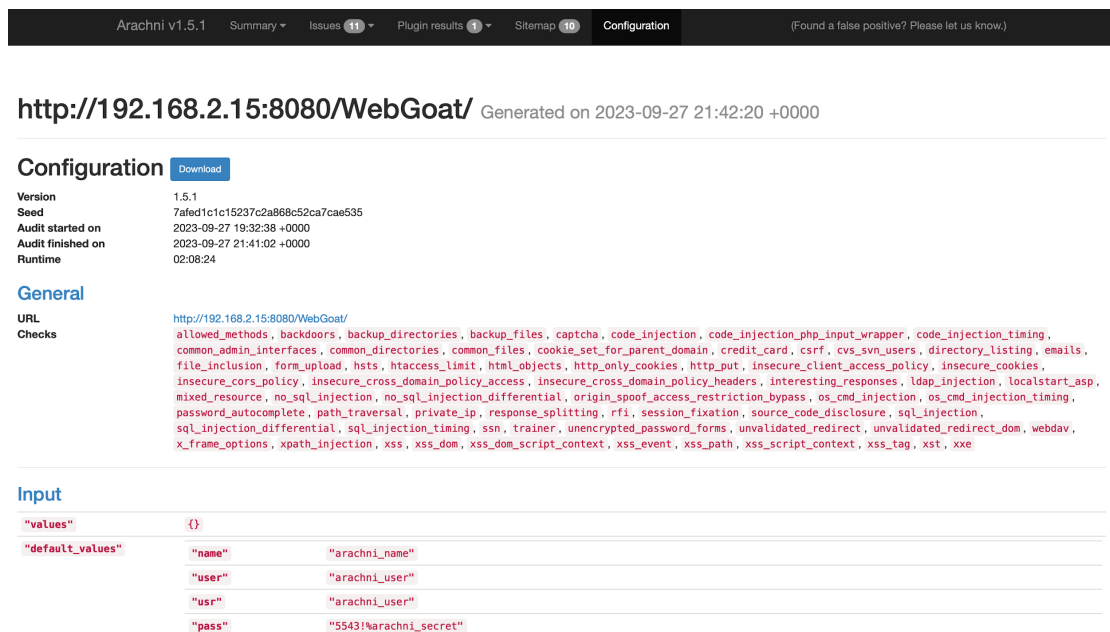


Figure 50: DAST Arachni report configuration

This structure facilitates users in effectively investigating and comprehending the security discoveries within the application, classifying them based on severity, trustworthiness, and pertinence to the OWASP Top Ten list. The report's user-friendliness is especially apparent when seen in a localized context, as it

combines visual elements such as colors and charts that improve the overall comprehension of the vulnerabilities identified throughout the scanning process.

Presented below is a tabular representation that succinctly outlines the advantageous and disadvantageous attributes of the Arachni tool, which is utilized for the purpose of conducting web application security screening:

Aspect	Positive Aspects	Negative Aspects
<b>Comprehensive Scanning</b>	Scans for a wide range of vulnerabilities, including XSS, SQL injection, and more.	High false-positive rate can lead to time-consuming manual verification.
<b>Automation</b>	Supports automation and integration with CI/CD pipelines for continuous scanning.	Initial configuration and setup can be complex.
<b>Customization</b>	Offers customization of scan profiles and reporting options to suit specific needs.	Extensive customization may require expertise.
<b>Reporting</b>	Generates detailed and structured reports, including HTML, JSON, and others.	Reports can be overwhelming for large scans.
<b>Interactive Reports</b>	Provides interactive HTML reports with clickable links and detailed information.	The volume of information may be challenging to navigate for less experienced users.
<b>Vulnerability Severity</b>	Categorizes vulnerabilities by severity, helping prioritize remediation efforts.	Vulnerability categorization may not always align with an organization's risk assessment.
<b>Community Support</b>	Active open-source community for updates, plugins, and bug fixes.	Limited official support compared to paid tools.
<b>Web Crawling</b>	Uses a crawler to discover and scan pages.	May not handle complex or dynamic content well.
<b>Authentication Support</b>	Can handle login forms and sessions for authenticated scans.	Configuring authentication can be challenging for complex authentication mechanisms.
<b>Resource Intensive</b>	Can be resource-intensive, especially for large and complex web applications.	Large scans may require substantial computing resources and time.

*Table 7: Arachni tool positives and negatives*

## Tools Comparison

### Comparison of SAST Tools SonarQube and SNYK

This analysis aims to examine the two used Static Application Security Testing (SAST) tools, SonarQube and SNYK, highlighting their respective merits and limitations. SonarQube is renowned for its extensive analysis of code quality and its ability to customize rules that enhance maintainability, dependability, and efficiency. In contrast, SNYK prioritizes security analysis, efficiently detecting vulnerabilities and classifying them based on their level of seriousness.

**Code Quality Analysis:** SonarQube provides a complete analysis of code quality, prioritizing the aspects of maintainability, dependability, and efficiency. On the contrary, SNYK places greater emphasis on security analysis and may not offer an extensive evaluation of code quality.

**Security Analysis:** The security analysis capabilities of SonarQube encompass the identification and mitigation of common vulnerabilities, namely those outlined in the OWASP Top 10. By utilizing SonarQube, developers may adopt safer coding practices. In contrast, SNYK primarily emphasizes efficient identification of vulnerabilities through its security analysis features.

**Custom Rules:** The SonarQube tool offers the ability to create custom rules for code quality and security, enabling businesses to customize their inspections according to their own needs. In contrast, SNYK does not provide an equivalent degree of customisation in terms of code quality and security standards.

**Integration:** SonarQube has a high degree of compatibility with diverse development environments, continuous integration/continuous deployment pipelines, and integrated development environments. SNYK exhibits effective integration capabilities, albeit necessitating preliminary configuration, a task normally undertaken only once.

**Interactive Dashboards:** SonarQube provides users with interactive and configurable dashboards that display code quality and security metrics. However, effectively utilizing these dashboards and comprehending the information presented may need a certain level of training and familiarity with the user interface. While SNYK does offer user-friendly dashboards, it may not give an equivalent level of code quality analysis.

**Scalability:** SonarQube has strong scalability in its capacity to analyze extensive and intricate codebases, albeit necessitating adequate hardware resources. However, it should be noted that SNYK has the potential to be effective in its code analysis. Nevertheless, it is important to consider that the scalability of SNYK may differ depending on the complexity of the codebase.

**Support for Multiple Languages:** SonarQube offers support for a wide range of programming languages, hence offering users increased freedom in their choice of language. While SNYK prioritizes security analysis across many programming languages, it may not provide a same level of code quality analysis for all supported languages.



**Continuous Monitoring:** Continuous monitoring is a crucial aspect of software development, as it enables the timely identification of code quality and security concerns. SonarQube, a widely used tool, offers this capability by providing ongoing monitoring throughout the development process. However, it is worth noting that setting up SonarQube may present some challenges. SNYK provides continuous security monitoring through the implementation of scheduled scans, albeit the regularity of these scans may result in a substantial influx of generated reports.

**Cost:** SonarQube provides a free Community Edition that encompasses fundamental functions, but more sophisticated features and support necessitate purchase. SNYK offers a range of pricing options, including both complimentary and premium plans, therefore accommodating diverse project needs and allowing for more adaptability. The expenses associated with large-scale or intricate projects may escalate due to the inclusion of supplementary features or services.

In summary, SonarQube has exceptional proficiency in delivering thorough code quality analysis, allowing for customization, and offering support for a wide range of programming languages. In contrast, SNYK places greater emphasis on the optimization of security analysis, seamless integration, and compatibility with GitHub. The selection of tools should be based on the project's individual objectives, taking into consideration factors such as the importance placed on code quality, security analysis, and integration demands.

### Comparison of Security Container Analysis Tools Grype and Trivy

Grype and Trivy are two tools specifically developed to augment container security through the identification of vulnerabilities present in container images and filesystems. This analysis will examine the fundamental characteristics and factors to be considered when comparing Grype and Trivy, elucidating their individual merits and constraints.

**Functionality:** Both Grype and Trivy demonstrate efficient capabilities in doing vulnerability scans on container images and filesystems.

Both methods are restricted to conducting scans for vulnerabilities in established software packages and libraries.

**Vulnerability Database:** The Vulnerability Database is a comprehensive repository of information pertaining to vulnerabilities in various software systems and applications.

**Container Support:** Grype and Trivy support various container image formats, including Docker and OCI images.

Both approaches largely prioritize containerized apps, possibly disregarding other assets.

**Severity Assessment:** The evaluation of severity in both tools is reliant on the data that is accessible and may not consistently provide an accurate representation of risk in real-world scenarios.

**Customization:** Both Grype and Trivy offer the capability to create custom rules in order to adapt the scanning process to meet unique requirements. The process of customization necessitates a comprehensive understanding and has the potential to result in inaccurate bad outcomes in both tools.

**Integration:** Grype and Trivy exhibit a high level of compatibility with continuous integration and continuous deployment (CI/CD) pipelines, hence facilitating the automation of processes and the adoption of DevSecOps methodologies. Both of these tasks necessitate exerting effort in order to properly install and seamlessly incorporate them into pre-existing operations.

**Open Source:** Both Grype and Trivy are open-source software, which means that they are openly accessible and allow for code examination and change. The quality of support and documentation for tools in community-supported software might vary.

**Report Generation:** Both tools have the capability to create comprehensive reports that provide a concise overview of vulnerabilities, facilitating convenient examination. The correctness of the vulnerability database may have an impact on the quality of the report generated by both Grype and Trivy.

**Resource Utilization:** Both tools are often characterized by their low weight and minimal impact on system resources during scanning processes. The utilization of resources in both tools is subject to variation according on the dimensions and intricacy of container pictures.

**Continuous Updates:** Both Grype and Trivy regularly update their vulnerability databases to assure comprehensive coverage of the most recent security threats. In order to ensure the continued efficacy of both technologies, it is imperative for users to actively engage in the process of updating the databases.

In brief, Grype and Trivy exhibit several commonalities with respect to their functionality, database management, container compatibility, severity evaluation, customization options, integration capabilities, open-source characteristics, report production, resource utilization, and ongoing upgrades. The selection between these options may ultimately be determined by certain criteria and preferences inside one's container security protocols.

### Comparison of DAST Tools Arachni and OWASP-ZAP

The Arachni vulnerability detection tool distinguishes itself by doing comprehensive scans to identify a diverse array of vulnerabilities, encompassing prevalent security risks such as Cross-Site Scripting (XSS) and SQL injection. Nevertheless, it is important to acknowledge that Arachni has the potential to produce false positives,

indicating the possibility of identifying issues as vulnerabilities when they are not, hence necessitating manual verification.

In contrast, OWASP ZAP demonstrates a high level of efficacy in detecting vulnerabilities within online applications, therefore establishing itself as a dependable option for doing security scans. Similar to Arachni, it is possible for this tool to generate false positives, which would require a manual review process.

**Integration Capabilities:** Arachni possesses the ability to automate processes and can be seamlessly included into Continuous Integration/Continuous Delivery (CI/CD) pipelines. However, several users perceive the initial setup and configuration as intricate, posing a potential obstacle for individuals who are new to the system.

OWASP ZAP is renowned for its seamless integration capabilities with widely adopted solutions such as Jenkins and Docker. The process of establishing and integrating the system with your current infrastructure is pretty uncomplicated. Nevertheless, the process of updating ZAP may include some cleansing procedures, which might be perceived as a disadvantage.

**Customization:** Arachni provides a wide range of customization options for configuring scan profiles and generating reports. Individuals have the ability to customize the tool according to their own needs and preferences. Nevertheless, the exploration of customisation at a profound level may necessitate a certain degree of knowledge, hence posing a problem for individuals with limited experience.

The OWASP ZAP tool possesses the capability to be customized according to unique testing settings, hence enabling users to modify its functionality to align with their individual requirements. However, in order to get best outcomes with ZAP, it may be necessary to possess a comprehensive comprehension of web application security, rendering it more ideal for those who possess knowledge in this domain.

Arachni is capable of generating comprehensive and organized reports in several forms, such as HTML and JSON. Although these reports offer extensive insights, they can become daunting when dealing with huge scans. Therefore, it is crucial to properly navigate through the data.

The ZAP tool offers comprehensive and organized results that include plugin IDs, facilitating the identification of vulnerabilities. Nevertheless, several users perceive the report viewing interface in Jenkins to possess a relatively lower level of user-friendliness, hence potentially constituting a slight disadvantage.

The concept of vulnerability refers to the state or condition of being susceptible to harm.

**Severity:** The categorization of vulnerabilities by Arachni assists businesses in prioritizing the resolution of issues based on their severity. This enables organizations to choose which vulnerabilities should be addressed first. This particular characteristic facilitates the implementation of effective remedial actions. Nevertheless, it is crucial to acknowledge that the classification of severity may not consistently correspond precisely with an organization's evaluation of risk.

The OWASP ZAP tool provides valuable information into the security stance of online applications, aiding users in gaining a deeper understanding of their vulnerabilities. Nevertheless, its value is contingent upon the user's pre-existing understanding of web application security, rendering it particularly beneficial for individuals who possess competence in this domain.

**Community Support:** Both Arachni and OWASP ZAP derive advantages from the presence of vibrant open-source communities that actively contribute to the development of updates, plugins, and bug patches. However, it is vital to consider that the extent of government support for these tools may be restricted in comparison to paid options.

**Resource Usage:** The utilization of resources by Arachni can be significant, particularly in the context of scanning online applications that are both extensive in size and intricate in nature. In resource-constrained situations, the completion of scans may necessitate significant computational resources and time, hence posing a potential challenge for users. The resource efficiency of OWASP ZAP is reasonable during scanning operations. However, when dealing with complicated applications or larger-scale testing, the use of resources may rise.

Open-source tools, such as Arachni and OWASP ZAP, are characterized by their availability for unrestricted usage and the transparency of their source code, allowing users to review and modify them as desired. The open-source nature of these technologies cultivates inclusive communities that offer many resources to users.

In brief, Arachni demonstrates proficiency in conducting thorough scans and offering customization options, but perhaps accompanied by a greater likelihood of false positives and an initial learning curve. The OWASP ZAP tool is renowned for its seamless integration capabilities, efficient dynamic analysis functionality, and well-organized reporting system. However, it is important to note that there is a possibility of false positives and a certain level of skill is necessary for its effective utilization. The selection between these options should be in accordance with one's particular requirements, level of proficiency, and concerns regarding infrastructure.

## Conclusion

This thesis explores the complexities of a multidimensional journey within the domain of DevSecOps, where the integration of software development and cybersecurity is of utmost importance. The primary objective of this research was to examine the implementation and incorporation of security protocols throughout the Software Development Lifecycle (SDLC), within an environment that is progressively shaped using DevOps methodologies.

The investigation began with acquiring a thorough comprehension of the Software Development Lifecycle, supported by a full examination of the Agile Manifesto, which serves as the fundamental principle of contemporary software development. Ultimately, we proceeded to explore the realm of Continuous Integration and Continuous Delivery (CI/CD) methodologies, ultimately encountering the transformative impact of DevOps. The major focus of this thesis is the Application Security Pipeline for Continuous Integration/Continuous Deployment (CI/CD), which is an area that combines Information Security and Application Security. This highlights the significant importance of security inside the DevOps framework.

As we proceeded farther, our attention was drawn to the weaknesses that became more prominent. The present study conducted a comprehensive analysis of several methodologies employed in the identification of vulnerabilities across the Software Development Life Cycle (SDLC). The study examined Static Code Analysis, Dynamic Code Analysis, Software Composition Analysis, and Container Security Analysis, which provided distinct perspectives on the security status of the scrutinized programs.

The central focus of this study revolves around the proposition and assessment of a Continuous Vulnerability Management framework within the context of DevSecOps. This study has presented an analysis of the practices involved in Continuous Vulnerability Evaluation, Treatment, and Reporting, emphasizing the need of using proactive and iterative approaches to vulnerability management.

In the pursuit of genuineness and verification, a comprehensive practical case study was conducted utilizing the OWASP WebGoat program. A comprehensive evaluation was conducted on various security technologies as part of a DevSecOps pipeline. These technologies included Static Application Security Testing (SAST) tools such as SNYK and SonarQube, Container Security Analysis tools like Grype and Trivy, as well as Dynamic Application Security Testing (DAST) tools including OWASP-Zap and Arachni. The objective of this study was to assess the effectiveness of these technologies, providing a substantial amount of valuable insights and data.

In summary, this thesis has effectively shown a significant correlation between software development and cybersecurity, providing valuable perspectives and recommendations for businesses seeking to strengthen their security protocols in the ever-evolving landscape of modern software development. The comprehensive methodology employed, which encompasses the entirety of the Software Development Lifecycle and incorporates the fundamental principles of DevSecOps, serves as a demonstration of the dedication to constructing software systems that are

both safe and resilient. The ongoing evolution of the digital ecosystem has prompted enterprises to seek significant insights and guidance from this research, which may help them succeed in a safe, DevOps-driven environment.

## Bibliography

- [1] "Aqua Security," [Online]. Available: <https://aquasecurity.github.io/trivy/dev/>.
- [2] Atlassian, "What Is DevOps?," [Online]. Available: <https://www.atlassian.com/devops>. [Accessed 2023].
- [3] D. Spinellis, "ieeexplore," 2016. [Online]. Available: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7458759>.
- [4] R. Jabbari, N. Bin All, B. Tanveer and K. Petersen, "What is DevOps?: A Systematic Mapping Study on Definitions and Practices.," 2016.
- [5] Red Hat, "What is a CI/CD pipeline?," 11 May 2022. [Online]. Available: <https://www.redhat.com/en/topics/devops/what-cicd-pipeline>. [Accessed 2023].
- [6] I. Ltd, Iso/iec 27034:2011+ -information technology -security techniques - application security.
- [7] R. N. Rajapaske, M. Zahedi, M. Ali Babar and H. Shen, "Challenges and Solutions When Adopting DevSecOps: A Systematic Review," *Information and Software Technology*, 2022.
- [8] OWASP, "OWASP top Ten," [Online]. Available: <https://owasp.org/www-project-top-ten/>.
- [9] A. Martin, S. Raponi, T. Combe and R. Di Pietro, "Docker ecosystem–vulnerability analysis," *Computer Communications*.
- [10] FORTINET, "FORTINET," [Online]. Available: <https://www.fortinet.com/resources/cyberglossary/threat-modeling>. [Accessed 2023].
- [11] "Secure Software Technologies," in *Information and System Security in the Cyberspace*, NewTech Pub, 2021.
- [12] OWASP, "OWASP," [Online]. Available: [https://owasp.org/www-community/controls/Static\\_Code\\_Analysis](https://owasp.org/www-community/controls/Static_Code_Analysis). [Accessed 2023].
- [13] ISECT LTD, "Iso/iec 27034:2011+ - information technology - security techniques - application security (all except part 4 published)," [Online]. Available: <https://www.iso27001security.com/html/27034.html>.
- [14] T. Nora, L. Jingyue and H. Huang, "An Empirical Study on Culture, Automation, Measurement, and Sharing of DevSecOps," *IEEE*, 3 June 2019.
- [15] T. Rangnau, R. v. Buijtenen, F. Fransen and F. Turkmen, "Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines," *IEEE*, 2020.
- [16] OWASP, "OWASP," [Online]. Available: [https://owasp.org/www-community/Source\\_Code\\_Analysis\\_Tools](https://owasp.org/www-community/Source_Code_Analysis_Tools).
- [17] OWASP, "Checkpoint," [Online]. Available: <https://www.checkpoint.com/cyber-hub/cloud-security/what-is-dynamic-code-analysis/>.
- [18] Stackify, "Stackify," [Online]. Available: <https://stackify.com/oop-concepts-composition/>.
- [19] Helpnetsecurity, "Helpnetsecurity," [Online]. Available: <https://www.helpnetsecurity.com/2020/01/22/container-security-continuous-security/>.
- [20] RedHat, "A comprehensive DevSecOps solution," [Online]. Available: <https://www.redhat.com/en/partners/devsecops>.

- [21] zscaler, “zscaler,” 18 May 2022. [Online]. Available: <https://www.zscaler.com/blogs/product-insights/top-challenges-faced-organizations-implementing-devsecops>. [Accessed 2023].
- [22] CISCO, “CISCO - DevSecOps - Addressing Security Challenges in a Fast Evolving Landscape White Paper,” 18 March 2021. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/devsecops-addressing-security-challenges.html>. [Accessed 2023].
- [23] IBM, “What is DevSecOps?,” [Online]. Available: <https://www.ibm.com/topics/devsecops>. [Accessed 2023].
- [24] snyk, “snyk,” [Online]. Available: <https://snyk.io/about/>. [Accessed September 2023].
- [25] “wikipedia,” [Online]. Available: <https://en.wikipedia.org/wiki/SonarQube>.
- [26] SonarQube. [Online]. Available: <https://docs.sonarsource.com/sonarqube/latest/>.
- [27] “GitHub,” [Online]. Available: <https://github.com/aquasecurity/trivy>.
- [28] “How-To-Geek,” [Online]. Available: <https://www.howtogeek.com/devops/how-to-use-trivy-to-find-vulnerabilities-in-docker-containers/>.
- [29] “Aqua Security,” [Online]. Available: <https://www.aquasec.com/products/open-source-projects/>.
- [30] Github.com. [Online]. Available: <https://github.com/anchore/grype>.
- [31] ZAP, “ZAP,” [Online]. Available: <https://www.zaproxy.org/getting-started/>.
- [32] “InfoSsc,” [Online]. Available: <https://resources.infosecinstitute.com/topics/application-security/web-application-testing-with-arachni/>.
- [33] “comparitech,” [Online].
- [34] “Network Admin Tools,” [Online]. Available: <https://www.comparitech.com/net-admin/what-is-dast/>.
- [35] “UpGuard,” [Online].
- [36] OWASP, “OWASP,” [Online]. Available: [https://owasp.org/www-community/Threat\\_Modeling](https://owasp.org/www-community/Threat_Modeling). [Accessed 2023].