

Nikesh Bahadur Adhikari

EVALUATING SECURITY TOOLS IN THE CONTEXT OF DEVSECOPS

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Zheyang Zhang
Marko Helenius
April 2024

ABSTRACT

Nikesh Bahadur Adhikari: Evaluating security tools in the context of DevSecOps
Master of Science Thesis
Tampere University
Master's Degree Programme in Computing Sciences - Software, Web & Cloud
April 2024

DevSecOps (Development, Security, and Operations) paradigm is simply an expansion of the DevOps (Development, and Operations), which intersects all the development, security, and operation activities under the same roof to develop a secure product continuously in an iterative approach. Recently, DevSecOps has started becoming a popular approach for most organizations. However, identifying the right security tools is always a concern in DevSecOps.

The primary objective behind this thesis is to determine the strengths and limitations based on general benchmarking and performance metrics of both Security Application Security Testing (SAST) and Infrastructure as Code (IaC) tools. We aspired to assist stakeholders interested in implementing the DevSecOps paradigm in Azure pipelines with a static analysis as a starting point. In addition, an attempt is made to map the OWASP Top 10 vulnerabilities with various security activities that detect them in DevSecOps.

We performed two case studies on two benchmarks in an Azure cloud environment, i.e., OWASP Juice Shop for SAST and Terragot for IaC tools. Semgrep, Sonarcloud, and Snyk were selected as SAST tools, whereas Checkov, Snyk IaC, and Tfsec were chosen as IaC tools. Each tool has its Azure pipeline setup for scan which was executed independently to collect the data of the scan results. The collected data was analyzed for performance metrics, while general metrics were collected from their official documentation.

The outcomes of general and performance metrics reveal that each tool has its strengths and limitations, and the selection of security tools can be done according to one's requirements. For instance, Semgrep stands out to be better in performance metrics in SAST analysis, while Snyk is the fastest in performing scans and SonarCloud has the largest programming language support for static analysis. Similarly, Tfsec is the easiest to get started for IaC scans. Snyk IaC has also a good number of updated Azure policies while Checkov is prominent in terms of performance due to its largest datasets of known Azure policies. On the other hand, OWASP's Top 10 vulnerabilities besides "A09:2021-Security Logging and Monitoring Failures" are detected by the SAST tools.

Keywords: DevSecOps, DevOps, SAST, IaC, security testing, code smell, OWASP Top 10 vulnerabilities, static analysis

The originality of this thesis has been checked using the Turnitin Originality Check service.

PREFACE

I want to express my gratitude to the factory automation company for supporting this thesis, and to all my colleagues for their helpful feedback and support. A special thank you to Tommi and Jani for guiding me and keeping me motivated. Also, I'm thankful to my professors Zheyang and Marko for their advice and constructive feedback during the process of writing the thesis.

Lastly, I want to give a big thank you to my family, uncle, and special friends for their encouragement, motivation, and support throughout my studies. I couldn't have accomplished this without each of you. Thank you so much.

Tampere, 1st April 2024

Nikesh Bahadur Adhikari

CONTENTS

1	INTRODUCTION	1
1.1	Research Objective.....	1
1.2	Thesis Structure.....	2
2	WEB VULNERABILITIES	3
2.1	Common Web Vulnerabilities.....	3
2.1.1	A01:2021-Broken Access Control	4
2.1.2	A02:2021-Cryptographic Failures	4
2.1.3	A03:2021-Injection.....	5
2.1.4	A04:2021-Insecure Design	5
2.1.5	A05:2021-Security Misconfiguration	6
2.1.6	A06:2021-Vulnerable and Outdated Components.....	6
2.1.7	A07:2021-Identification and Authentication Failures.....	7
2.1.8	A08:2021-Software and Data Integrity	7
2.1.9	A09:2021-Security Logging and Monitoring Failures	7
2.1.10	A10:2021-Server-Side Request Forgery (SSRF)	8
2.1.11	Summary	8
2.2	Common Security Smells in Infrastructure as Code (IaC)	9
2.2.1	Administration by Default.....	10
2.2.2	Empty Password	10
2.2.3	Hard-coded Secret.....	11
2.2.4	Unrestricted Internet Protocol (IP) address binding.....	11
2.2.5	Missing Default in Case Statement.....	11
2.2.6	No Integrity Check.....	11
2.2.7	Suspicious Comment.....	11
2.2.8	Using HTTP without Transport Layer Security (TLS).....	12
2.2.9	Using Weak Cryptography Algorithm	12
2.3	Mapping of IaC Security Smells with MITRE CWE and OWASP Top 10 Vulnerability	12
3	SECURITY TESTING AND TECHNIQUES	14
3.1	Security Testing	15
3.2	Types of Security Analysis Tools in DevSecOps.....	16
3.2.1	Static Application Security Testing (SAST) Tools.....	16
3.2.2	Dynamic Analysis Security Testing (DAST) Tools.....	17
3.2.3	Interactive Application Security Testing (IAST) Tools	18
3.2.4	Software Composition Analysis (SCA) Tools.....	19
3.2.5	Infrastructure Vulnerability Scanning Tools	20
3.2.6	Container Vulnerability Scanning Tools	20
4	DEVOPS AND DEVSECOPS	22
4.1	DevOps - Development and Operations.....	22
4.1.1	DevOps Dimension	22
4.1.2	DevOps Cycle	23
4.1.3	DevOps Pipeline and Tools	25
4.1.4	Benefits and Shortcomings	27
4.2	DevSecOps – Development, Security and Operation.....	28
4.2.1	DevSecOps Principles	28
4.2.2	DevOps Vs DevSecOps Pipeline.....	29
4.2.3	DevSecOps Tools.....	31
4.2.4	Mapping DevSecOps Security Testing Activities with OWASP Top 10 Vulnerabilities	32

4.2.5	<i>Benefits and Challenges</i>	33
4.3	Related Works	33
5	METRICS OF ANALYSIS AND STUDY DESIGN	36
5.1	Tools Selection Criterion and Evaluation Metrics	36
5.1.1	<i>Tools Selection Criterion</i>	36
5.1.2	<i>Tool Evaluation Metrics</i>	37
5.2	Selected Tools	40
5.2.1	<i>SonarCloud</i>	40
5.2.2	<i>Semgrep</i>	41
5.2.3	<i>Snyk</i>	41
5.2.4	<i>Tfsec</i>	41
5.2.5	<i>Checkov</i>	42
5.3	Study Design Specification	43
5.3.1	<i>Selected Benchmarks</i>	43
5.3.2	<i>Test Environment</i>	45
5.3.3	<i>Data Collection</i>	48
6	RESULTS AND DISCUSSION	49
6.1	SAST Tools	49
6.1.1	<i>Vulnerability Detection and Metrics</i>	49
6.1.2	<i>Summary</i>	52
6.2	IaC Tools	55
6.2.1	<i>Vulnerability Detection and Metrics</i>	56
6.2.2	<i>Mapping of the Matched Vulnerabilities in Tool Comparison</i>	59
6.2.3	<i>Summary</i>	61
7	CONCLUSION	63
8	REFERENCES	65
9	APPENDICES	74
9.1	APPENDIX A: GROUND TRUTHS PRESENT IN OWASP JUICE SHOP	74
9.2	APPENDIX B: GROUND TRUTHS PRESENT IN TERRAGOAT	75
9.3	APPENDIX C: AZURE YAML CONFIGURATION FILES	78

LIST OF FIGURES

Figure 1. Stages of DevOps Cycle, adapted from (Yarlagadda & Teja, 2021)	24
Figure 2. Stages of DevOps CI/CD pipeline (Anastasov, 2022)	25
Figure 3. The gray square represents DevOps pipelines while purple indicates additional security activities in the DevSecOps pipeline, (Lombardi & Fanton, 2023). 30	
Figure 4. High-level configuration of the test environment	45
Figure 5. SAST tools' vulnerability detection in OWASP Juice shop	49
Figure 6. TP, FP, FN, and TN detection in OWASP Juice Shop	50
Figure 7. SAST tools' metrics comparison.....	51
Figure 8. IaC tools' vulnerability detection in Terragoat	56
Figure 9. Azure IaC rules triggered by each IaC tool.....	56
Figure 10. True Positive and False Negative obtained by IaC tools	57
Figure 11. IaC tool's metrics comparison	58

LIST OF TABLES

Table 1. Mapping of security smells with its corresponding MITRE’s CWE and potential OWASP Vulnerability, extension of (Rahman & Williams, 2021), (OWASP, 2021).....	13
Table 2. DevOps pipeline stages vs DevSecOps pipeline stages	29
Table 3. DevSecOps stages and its security activities and tools	31
Table 4. Correlation between DevSecOps security activities and the OWASP Top 10 vulnerabilities	32
Table 5. Related work key points comparison.....	35
Table 6. Summary of the selected SAST and IaC tools.....	42
Table 7. General benchmarking metrics for SASTs Tools.....	53
Table 8. OWASP categories detection by the SAST Tools.....	54
Table 9. Mapping of found vulnerabilities in Terragoat with Checkov Id, Tfsec Id, and Snyk Id, extension of (Bridgecrewio, 2023)	60
Table 10. General metrics comparison of IaC tools	61

ABBREVIATIONS

ASD	Agile Software Development
BRS	Business Requirement Specification
CCA	Cause-consequence analysis
CI/CD	Continuous Integration and Continuous Delivery
CVE	Common Vulnerabilities and Exposures
CWE	Common weakness enumeration
DAST	Dynamic Application Security Testing
DevOps	Development and Operations
FN	False Negative
FP	False Positive
FTA	Fault tree analysis
HTTP	Hypertext Transfer Protocol
IaC	Infrastructure as a Code
IAST	Interactive Application Security Testing
MBST	Model Based Security Testing
MBT	Model Based Testing
NVD	National Vulnerability Databases
OSS	Open Source Software
OWASP	Open Worldwide Application Security Project
QA	Quality Assurance
RBAC	Role Base Access Control
SaaS	Software as a service
SAST	Static application security testing
SCA	Software composition analysis
SDLC	Software Development Life Cycle
SQL	Structure Query Language
SRS	Software Requirement Specification
SUT	System Under Test
TN	True Negative
TP	True Positive
UAT	User Acceptance Testing
VM	Virtual Machine
WAF	Web application firewalls
WAVSEP	Web Application Vulnerability Security Evaluation Project
WIVET	Web Input Vector Extractor Teaser

1 INTRODUCTION

Development and Operations (DevOps) practices have effectively integrated agile software development methodologies with operational teams throughout the software product lifecycle. The main objective is to connect software product development and deployment activities for faster and more efficient delivery. (Mao et al., 2020)

Nevertheless, the integration of security testing has fallen behind, often handled in a traditional waterfall manner. To overcome this shortcoming, the Development, Operation, and Security (DevSecOps) model was introduced. It incorporates different security activities into various stages concerning software development and deployment lifecycle. This paradigm put the spotlight on the “shift-left” procedure toward security, which means integrating automated security measures starting from the planning and development phases. Additionally, DevSecOps distributes security responsibilities across all team members. This security activity ensures that there is an earlier detection of vulnerabilities so that developers have more time to address and rectify these issues promptly. In this context, choosing the security tools is always crucial to fully realize the benefits of the DevSecOps principles and automate security activities. (Rajapakse et al., 2022)

1.1 Research Objective

Integrating security tools in DevSecOps comes with the pain of selecting the right tools whether it is Commercial or Open-Source Software (OSS) from the enormous available lists in the market. In addition, the lack of comprehensive comparison among the available security tools in the context of Azure Cloud can leave us in a state of uncertainty about the tools.

The existing research mainly focuses on the most popular Static Application Security Testing (SAST) tools, often overlooking their assessment with modern web applications on cloud platforms and including Infrastructure as Cloud (IaC) tools comparison. Intending to address this shortcoming, our research discovers and assesses the effectiveness of SAST tools that have support for at least Python (Python, 2024), JavaScript (MDN Web Docs, 2024), Typescript (TypeScript, 2024), and C# (Microsoft, 2024) languages, while IaC tools with Terraform (HashiCorp, 2024) support only. Consequently, SAST scans are performed against the OWASP Juice Shop (2023) – Angular (2024) and Nodejs (2024)

and IaC scans with Terragoat (Bridgecrewio, 2023). Both are the goat projects with the existence of known vulnerabilities. With this evaluation, we seek to identify the most effective security tools for static analysis of the source and IaC code base along with the detection capabilities in terms of OWASP Top 10 vulnerabilities for SAST tools. The primary research question for this research is:

- R1 What are the strengths and limitations of different SAST and IaC security tools in the context of Azure DevSecOps?

Overall, this research utilizes the quantitative research methodology and aims to provide organizations with valuable perceptions about the selection and integration of security tools, enabling them to make effective decisions on enhancing the security of their CI/CD pipelines.

1.2 Thesis Structure

This thesis provides a comprehensive evaluation of the security tools, DevOps and DevSecOps, beginning with an introduction in Chapter 1 that outlines the general concepts, problems, research objectives and questions. Chapter 2 summarized common web vulnerabilities recognized by the Open Worldwide Application Security Project (OWASP), security smells in Infrastructure as Code (IaC), and their correlation with Common Weakness Enumeration (CWE) and OWASP vulnerability categories. Furthermore, in-depth details about security testing and tools utilized for security analysis in the DevSecOps paradigm are explained in Chapter 3 named Security Testing and Techniques. Chapter 4 offers a deep dive into the DevOps and DevSecOps paradigms, discussing their principles, pipelines, and the implementation of security tools at various software development stages. Study design details are presented in Chapter 5, where the methods, metrics, and test environment for evaluating Static Application Security Testing (SAST) and IaC tools are described, along with the criteria for their selection. In Chapter 6, the results and discussion highlight insights based on metrics from scans performed on selected benchmark projects. Finally, Chapter 7 concludes the thesis by summarizing the objectives, findings, contribution, limitations, and suggesting future improvements. Bibliographic references and appendix can be found respectively after Chapter 7 at the end of this thesis.

2 WEB VULNERABILITIES

Most software companies have adopted Software as a service (SaaS) as their offering software product in recent years, where software is provided to consumers through the Internet (Ibrahim et al., 2022). During the development of these applications, flaws and errors may emerge resulting in vulnerabilities. A vulnerability refers to a flaw or weakness in an application, which malicious users can exploit to obtain unauthorized access and compromise both the security and integrity of an application (Rahman & Williams, 2021). As these vulnerabilities persist in SaaS applications can be exploited easily leading to web-based attacks and security breaches (Elsayed & Zulkernine, 2019).

On the other hand, most SaaS application's infrastructure and configuration are managed, deployed, and orchestrated in an automated way using Infrastructure as a Code (IaC). IaC is merely software code that is prone to errors and defects, therefore leading to security issues as well. (Chiari et al., 2022)

Collectively, these cloud-based web applications might contain numerous flaws in their code and configuration, which makes them susceptible to various malicious cyber attacks (Mateo Tudela et al., 2020).

2.1 Common Web Vulnerabilities

Firstly, Common Vulnerabilities and Exposures (CVE) is the public catalog containing known software vulnerabilities. Each vulnerability in this catalog is assigned a distinct and unique identifier. While Common Weakness Enumeration (CWE) provides an overall description of the software flaws and weaknesses. CVEs are the identifiers for vulnerabilities, whereas CWEs describe the underlying weaknesses or flaws that lead to those vulnerabilities. (Sane, 2021) The list of both CVE and CWE is managed by the Massachusetts Institute of Technology Research and Engineering (MITRE) Corporation (MITRE Corporation, 2024).

The Open Worldwide Application Security Project (OWASP) is an open-source project started by a non-profit OWASP foundation organization in 2001 and founded in 2004 to enhance the security of software. This project is concerned more with web application

security and has recognized the top 10 critical vulnerabilities for the web as OWASP Top 10 2021. (OWASP, 2021)

In addition, OWASP Top 10 is recognized worldwide as a collection of critical security vulnerabilities for web applications and also serves as a benchmark for evaluating the security aspects of web applications. The in-depth details of those top 10 vulnerabilities including their corresponding Common Weakness Enumeration (CWE) that leads to those vulnerabilities are provided below. (OWASP, 2021)

2.1.1 A01:2021-Broken Access Control

Access control defines the policies about how a user can access or perform certain action on the resources and functionalities. In other words, it restricts the user's access control to those resources and functionalities according to permission defined in policies. Failure in access control restrictions will lead users to have an unauthorized information disclosure, also known as Broken Access Control. This is a most common security vulnerability as access control policies are defined by humans, making them highly susceptible to errors. (Aljabri et al., 2022)

URL and JSON Web Token tampering, exploiting CORS misconfigurations, invalidating API requests, ignoring the implementation of RABC for access resources, and storing sensitive user information on the client side are some of the most prevalent flaws that led to an access control vulnerability (OWASP, 2021). In addition, there are 34 CWE mapped to this vulnerability among which "CWE-201: Insertion of Sensitive Information Into Sent Data", "CWE-352: Cross-Site Request Forgery" and "CWE-200: Exposure of Sensitive Information to an Unauthorized Actor" are the most common ones. (OWASP, 2021)

2.1.2 A02:2021-Cryptographic Failures

The confidential information of users such as personal information, passwords, credit cards, and so on should be encrypted at all times whether the data is transmitted through a network or resides in some databases (Aljabri et al., 2022). Sensitive information might be exposed because of cryptographic vulnerabilities such as the use of outdated or weak encryption algorithms, weak transport layer security, weak SSL/TLS protocols, missing

HTTP header security directives, and hardcoded passwords. Such flaws allow for security breaches, also referred to as cryptographic failures. (OWASP, 2021)

Additionally, “CWE-327: Broken or Risky Crypto Algorithm”, “CWE-331 Insufficient Entropy” and “CWE-259: Use of Hard-coded Password” are notable Common Weaknesses Enumerations (CWEs) among 29 found in this vulnerability category (OWASP, 2021).

2.1.3 A03:2021-Injection

When the validation for the input data is omitted in a web application, it will open the door to an injection attack allowing untrusted input data to be processed by the interpreter. This attack tricks the interpreter into running the compromised input resulting in the exposure of sensitive data or allowing them to run the malicious code on both the client and server side. (Bach-Nutman, 2020) SQL injection, Code injection, and Lightweight Directory Application Protocol (LDAP) injection are some common examples of injection attacks (OWASP, 2021).

In addition, this vulnerability category is associated with 33 distinct Common Weakness Enumerations (CWEs), where “CWE-79: Cross-site Scripting”, “CWE-89: SQL Injection”, and “CWE-73: External Control of File Name or Path” are the common ones (OWASP, 2021).

2.1.4 A04:2021-Insecure Design

Design and architectural flaws can make an application susceptible to attacks. When security requirements, secure design patterns, threat modeling, error handling, and budget as well as time for including security activities are not considered during the design phase then, insecure design vulnerabilities might exist. Application exploitation can be avoided if it is developed with security in mind from the design phase. Because it helps to detect potential attack threats and foresee failure scenarios. Unhandled error exceptions and stack traces might lead to path transversal which is one of the vulnerabilities attacks caused by insecure design. (Aljabri et al., 2022)

Furthermore, “CWE-501: Trust Boundary Violation”, “CWE-209: Generation of Error Message Containing Sensitive Information”, “CWE-522: Insufficiently Protected Credentials” and “CWE-256: Unprotected Storage of Credentials” are the most occurred ones among 40 mapped CWEs for this vulnerability category (OWASP, 2021).

2.1.5 A05:2021-Security Misconfiguration

The application framework itself and its infrastructure need configuration to be hosted on the internet. Security misconfiguration such as missing HTTP security headers, misconfigured permissions, unpatched vulnerabilities, default configuration, unprotected files and directories, verbose error messages and traces, use of unnecessary features, etc. can happen in a web application. These misconfiguration vulnerabilities can be an entry point for compromising an application. (Patni & Vaidya, 2019)

Moreover, “CWE-16 Configuration and CWE-611 Improper Restriction of XML External Entity Reference” are the most notable CWEs among 20 CWEs mapped for this security misconfiguration vulnerability (OWASP, 2021).

2.1.6 A06:2021-Vulnerable and Outdated Components

Modern application integrates with various components and dependencies such as frameworks, libraries, extensions, plugins, packages, and additional software modules. The presence of vulnerabilities, whether known or unknown within these components is highly likable. There can be efforts dedicated to addressing these vulnerabilities through a series of fixes and patches. However, it’s vital to note that support for this continuous improvement and fixes may cease in certain component scenarios. Consequently, the usage of outdated components or those with known vulnerabilities, even if they function appropriately in their intended context, could potentially expose our application to security risks. (Aljabri et al., 2022)

“CWE-1104: Use of Unmaintained Third-Party Components” is one the most notable CWEs among the three mapped CWEs for this vulnerability (OWASP, 2021).

2.1.7 A07:2021-Identification and Authentication Failures

Most of the application usage privileges are based on the user's digital identity verified through the authentication process. Thus, it has become a crucial functionality in an application to avoid unnecessary unauthorized access. However, failure in identification and authentication exposes the application to threatening users to conduct different malicious activities. They might be able to compromise passwords, key and session tokens, or tamper authentication requests to hijack the other's user credentials against the application. In addition, the usage of weak passwords and no rate limits on login attempts become vulnerable to automated logic attacks to bypass the authentication. (Aljabri et al., 2022)

Among the 22 CWEs mapped for this category, the most common ones are “CWE-297: Improper Validation of Certificate with Host Mismatch”, “CWE-384: Session Fixation” and “CWE-287: Improper Authentication” (OWASP, 2021).

2.1.8 A08:2021-Software and Data Integrity

Modern application architecture incorporates components, libraries, and plugins from different external origins. Consequently, the risk of data and software being altered by malicious parties leading to sensitive data exposure and losses is high. Usage of untrusted components, libraries, and modules from external sources, insecure CI/CD pipeline configuration enabling unauthorized access, and insecure pipeline deployment, consisting of auto-update functionality without integrity verification are some examples that cause software and data integrity failures. (OWASP, 2021)

“CWE-494: Download of Code Without Integrity Check”, “CWE-829: Inclusion of Functionality from Untrusted Control Sphere” and “CWE-502: Deserialization of Untrusted Data” are a few of the notable Common Weakness Enumerations (CWEs) among ten mapped for this category (OWASP, 2021).

2.1.9 A09:2021-Security Logging and Monitoring Failures

Whenever an important incident happens such as login attempts, access control violations, or validation errors, it is important to have proper logging of these activities. So, the logging mechanism assists in effective monitoring, altering, and detecting suspicious security incidents occurring in an application. Therefore, insufficient or failure to log and

monitor makes it impossible to detect suspicious activities and further motivates attackers to initiate future attacks. (Bach-Nutman, 2020)

In this vulnerability category, the scope of CWE-778, which addresses Insufficient Logging, and encompasses additional weaknesses such as “CWE-223 - Omission of Security-relevant Information”, “CWE-532- Insertion of Sensitive Information into Log File” and “CWE-117 - Improper Output Neutralization for Logs” (OWASP, 2021).

2.1.10 A10:2021-Server-Side Request Forgery (SSRF)

Server-side request forgery (SSRF) vulnerabilities allow unauthorized requests from the server to the attacker’s unintended destination. Basically, in an SSRF attack, the attacker manipulates the input field or parameter to perform an unauthorized action such as making an HTTP request to a harmful remote location, port scanning, network scanning, and so on. This can result in accessing or modifying the internal resources, exposure of sensitive data, and performing harmful actions on the application. (Aljabri et al., 2022) There aren’t many CWEs mapped for this vulnerability category.

2.1.11 Summary

In a nutshell, the OWASP complies with the most significant security web vulnerabilities category. These vulnerabilities are common in web applications and frequently exploited by malicious users to compromise the security of an organization. In addition, OWASP's top 10 vulnerabilities are considered as a benchmark for evaluating the security of web applications. Furthermore, OWASP's top 10 project provides guidelines, examples, and best security practices for preventing different vulnerabilities. This assists the developers in developing an application with security in mind and omits the most common vulnerabilities in an application. An open-source project OWASP Top 10 which has been updated regularly since 2003 according to the evolving landscape of web applications in terms of security. (OWASP, 2021)

The most frequently encountered vulnerability category in most web applications, as per OWASP, is A01:2021-Broken Access Control. Weak authentication and session management or misconfigured roles with higher privileges can lead to unauthorized access causing security breaches in the system. The use of strong passwords and proper session

management, proper error logging, implementing Role Base Access Control (RBAC), and rate limit API are some approaches to prevent Broken Access Control vulnerability. (OWASP, 2021)

Of course, these vulnerabilities listed provided by the OWASP are not the only vulnerabilities category. The Common Weakness Enumeration (CWE) from SANS also provides the top 25 software errors, with most of these falling under OWASP's vulnerability category. Additionally, MITRE also offers a wide range catalog of vulnerability categories, where OWASP top 10 is a subset of this MITRE vulnerability catalog (Kim & Vouk, 2014).

We will emphasize more on OWASP vulnerability categories because they are specially designed to assess web applications in terms of security aspects. The next section includes security weaknesses that are commonly found in Infrastructure as Code (IaC). This approach allows us to identify vulnerabilities that are found in the IaC, a critical area in modern web application development and deployment.

2.2 Common Security Smells in Infrastructure as Code (IaC)

Rahman & Williams (2021) defines "Code smells are recurring coding pattern that indicates potential maintenance problems". This introduces us to the idea of security smells which are a subset of the code smells. In other words, security smells are the red flags in the code patterns that indicate underlying possible issues in the application code-base that may result in security breaches (Rahman & Williams, 2021). Security smells arise when the developer fails to comply with the design principles and standards.

Security smells and vulnerabilities are not the same thing, though, as security smells indicate possible weaknesses in a coding pattern, whereas vulnerabilities are actual flaws or weaknesses that could jeopardize an application's security. Therefore, security smells are a potential indicator of vulnerability but not all vulnerabilities are security smells. (Rahman & Williams, 2021)

For instance, a common security concern in IaC code is the presence of hard-coded secrets like passwords. If there's uncertainty regarding whether the password is included in the code then it serves as a warning sign requiring an inspection, known as a security smell. If the malicious user manages to access and somehow exploits that password, then it can become a real vulnerability (Rahman & Williams, 2021). Therefore, security smells need a serious inspection, as it can potentially lead to a vulnerability.

Infrastructure as Code (IaC) is defined as a method of managing and provisioning cloud infrastructure rapidly and reliably at scale through code instead of manual processes (Chiari et al., 2022). It follows DevOps methodology and uses descriptive model language to define and deploy cloud infrastructure such as virtual machines, networks, and so on (Chiari et al., 2022). It can be challenging to detect and analyze security smells in IaC, even though it makes infrastructure deployment more consistent. As discussed, security smells are potential problems and security risks in a code that needs to be fixed (Reddy Konala et al., 2023). The most common nine security smells found in the IaC are described below.

2.2.1 Administration by Default

This smell is about always setting the default user as an admin which has a higher level of privilege in terms of the right to a system (Chiari et al., 2022). It violates the 'principle of least privilege property' as users gaining higher privileges in terms of rights make a system vulnerable to attack. These security smells indicate "CWE-250: Execution with unnecessary privileges" weakness. Therefore, it's a best practice to assign a minimum number of necessary rights as far as possible to a user. (Rahman & Williams, 2021)

2.2.2 Empty Password

This smell occurs when a password field in the IaC script is left empty or set to null. It indicates the security weakness as it opens a door for malicious individuals to have unauthorized access to the system. For instance, if a remote Structure Query Language (SQL) server has the "root" user and an empty password, then anyone can access that server with full privileges without having to enter a password. The corresponding CWE for this security smell is "CWE-258: Empty password in configuration file". (Rahman & Williams, 2021)

2.2.3 Hard-coded Secret

When sensitive information such as username, password, cryptography keys, tokens, etc are included in the IaC script then, this security smell is known as a Hard-coded secret. This might expose secrets to the attackers allowing them to gain illicit access and compromise the system security. This security smell belongs to CWE named “CWE-798: Use of hard-coded credentials”. (Rahman & Williams, 2021)

2.2.4 Unrestricted Internet Protocol (IP) address binding

When the computing resources or server is assigned with the address 0.0.0.0, then it allows having a connection from any Internet Protocol (IP) address on the network. 0.0.0.0 means it can accept requests from any IP address within the same network. These security smells can expose resources or servers to unwanted malicious traffic. This security smell is represented by “CWE-284: Improper access control”. (Rahman & Williams, 2021)

2.2.5 Missing Default in Case Statement

This security smell often occurs when we forget to handle the default case in the case-conditional logic such as the switch statement. This becomes vulnerable as attackers might take advantage of unhandled scenarios leading them to provide the sensitive verbose error message through stack traces and faults. “CWE-478: Missing default case in switch statement” is the CWE entry associated with this category. (Rahman & Williams, 2021)

2.2.6 No Integrity Check

This security smell hands align with Common Weakness Enumeration named “CWE-353: Missing support for integrity check”. When the IaC script doesn’t verify the integrity of the downloaded packages or files, then “no integrity check” security smells might occur. No integrity check allows the malicious user to tamper with the downloaded packages or files which leads the system to be compromised. (Rahman & Williams, 2021)

2.2.7 Suspicious Comment

Suspicious comment security smells occur when there is a comment about a potential security issue, flaws, or defects. “TODO”, “FIXME” and “HACK” are some of the

keywords in the script that might indicate suspicious comments and reveals more information about security flaws or insecure coding practices. Moreover, this security smell is connected to weakness “CWE-546: Suspicious comment”. (Rahman & Williams, 2021)

2.2.8 Using HTTP without Transport Layer Security (TLS)

HTTP communication without TLS is unsecured and including it on the IaC script is considered as a security smell. The usage of HTTP allows the attacker to intercept and tamper the request which makes the system vulnerable to attack and increases the risk of compromising sensitive information. This smell is associated with “CWE-319: Cleartext transmission of sensitive information”. (Rahman & Williams, 2021)

2.2.9 Using Weak Cryptography Algorithm

The usage of cryptography algorithms that are deprecated or outdated for encrypting confidential data such as passwords etc. in IaC script is known as “Using weak cryptography algorithm” security smells. The sensitive information protected by weak cryptography such as the Secure Hash Algorithm (SHA-1) and Message-digest Algorithm (MD5) can be exploited easily by the malicious attacker to tamper with the information or gain unauthorized access. A Common Enumeration Weakness entry named “CWE-326: Inadequate encryption strength” is connected to this security smell. (Rahman & Williams, 2021)

The above lists are the common security smells that are found in the IaC code which needs inspection as it could indicate a weakness that would raise a vulnerability in an application. The following section connects the security smells in IaC code to the potential OWASP vulnerability that could result from them.

2.3 Mapping of IaC Security Smells with MITRE CWE and OWASP Top 10 Vulnerability

Table 1 in our study associates the security smells discovered in Infrastructure as Code (IaC) code with both MITRE’s Common Weakness Enumeration (CWE) and OWASP’s top 10 vulnerability category. Rahman & Williams (2021) have established the

connection of security smell with its respective CWE weakness. Additionally, we extended the mapping of CWEs of security smell with their corresponding CWEs in OWASP's top 10 vulnerabilities, enabling the identification of the OWASP vulnerability category. The outcome of this mapping revealed that most of the security smells found on IaC code can potentially lead to the vulnerabilities listed in the OWASP top 10.

Table 1. Mapping of security smells with its corresponding MITRE's CWE and potential OWASP Vulnerability, extension of (Rahman & Williams, 2021), (OWASP, 2021)

Security Smell	MITRE's Common Weakness Enumeration (CWE)	OWASP Top 10 Vulnerability
Admin by default	CWE-250: Execution with unnecessary privileges	A05:2021-Security Misconfiguration
Empty password	CWE-258: Empty password in the configuration file	None
Hard-coded secret	CWE-798: Use of hard-coded credentials	A07:2021-Identification and Authentication Failures
Unrestricted Internet Protocol (IP) address binding	CWE-284: Improper access control	A01:2021-Broken Access Control
Missing default in case statement	CWE-478: Missing default case in switch statement	None
No integrity check	CWE-353: Missing support for integrity check	A08:2021-Software and Data Integrity Failures
Suspicious comment	CWE-546: Suspicious comment	None
Using HTTP without Transport Layer Security (TLS)	CWE-319: Cleartext transmission of sensitive information	A02:2021-Cryptographic Failures
Use of weak cryptography algorithm	CWE-326: Inadequate encryption strength	A02:2021-Cryptographic Failures

In summary, some OWASP top 10 vulnerabilities such as A08:2021-Software and Data Integrity failures, A02:2021-Cryptographic Failures, A01:2021-Broken Access Control, and A05:2021-Security Misconfiguration have the CWE weakness associated with security smells in their mapped CWE list. However, three security smells with CWE identifiers such as “CWE-258: Empty password in configuration file”, “CWE-478: Missing default case in switch statement” and “CWE-546: Suspicious comment” did not belong to any OWASP vulnerabilities mapped CWE entries.

3 SECURITY TESTING AND TECHNIQUES

Software testing is the process of validating whether software performs the intended functionality as expected or not. The primary motive is to discover the error as far as possible for software testing. Errors and faults are not always detected by testing and testing every aspect is also not feasible (Uddin & Anand, 2019). The software functionality might be affected by the errors which are classified and fixed according to the severity of that error. This is a continuous process in agile software development until no more errors are detected. (Taley & Pathak, 2020) In addition, testing also validates and verifies that software is defect-free and aligned with the user's requirements. As a result, it assists in maintaining the quality of software. (Uddin & Anand, 2019)

Uddin & Anand (2019) stated the main objectives of software testing as follows:

- Identifying and preventing defects.
- Meeting the requirements of the Software Requirement Specification (SRS) and Business Requirements Specification (BRS).
- Writing test cases with higher quality.
- Estimating the reliability of software.
- Optimal cost and minimal effort.
- Building trust with customers.

In addition, the ISO 25000 series specifies essential criteria that can be used to test the software quality (Ebert et al., 2022). These criteria include functional suitability, security, maintainability, usability, and performance efficiency. In functional suitability criteria, software should perform what it is intended to do as per the specified requirement (Ebert et al., 2022). Security should ensure that sensitive data is protected in software (Ebert et al., 2022). Maintainability describes the characteristics of software that should be flexible toward changes (Ebert et al., 2022). Usability ensures that software should guarantee the quality of user experience which is measured by effectiveness, efficiency, and satisfaction while using the software product from users (Ebert et al., 2022). Finally, performance efficiency criteria describe how well a system performs with its resources under different conditions (Ebert et al., 2022).

In general, software needs to be tested in terms of the above aspects to ensure quality. Additionally, security testing is one of the important activities that is carried out to protect software data against unauthorized access (Ebert et al., 2022).

3.1 Security Testing

Security is always a concern for any software in the market. In the context of a web application, we should prioritize security because they are publicly accessible and become more vulnerable to malicious attacks. Over the years, many techniques, tools, and frameworks have been introduced and suggested by experts for accessing web application security. Web application typically includes client-side and server-side scripts which can be written in different languages. The client and server communicate using Hypertext Transfer Protocol (HTTP). As web applications become complex and diverse, ensuring their security becomes vital for achieving success in the business and organization. (Aydos et al., 2022)

According to Felderer et al. (2016), security properties are defined as “Confidentiality, Integrity, Availability, Authentication, Authorization, and Non-repudiation”. Confidentiality ensures that sensitive information won’t be shared with unauthorized users. Integrity is such that the information or data remains accurate, consistent, and unaltered throughout the lifecycle and prevents unauthorized tampering with data. Availability ensures that authorized users consistently have access to the data. Authentication is a security approach that verifies the user’s identity. Authorization is also a security measure that grants access rights to the verified user. Lastly, Non-repudiation guarantees that none of the parties involved in a transaction may subsequently reject their participation. (Felderer, et al., 2016)

Security testing primarily focuses on detecting vulnerabilities and making sure that an application is safe from any malicious threat. Security functional testing and security vulnerability testing are the two different categories of security testing (Aydos et al., 2022). Security functional testing emphasizes validating that the security aspects required for the software are implemented and aligned with required security requirement specifications (Aydos et al., 2022). In contrast, uncovering possible security vulnerabilities from the

attacker's perspective is the main aim of security vulnerability testing (Aydos et al., 2022).

The OWASP outlines valuable web application testing guides to enhance application security. It incorporates various resources like OWASP's top 10 vulnerabilities, a guide for code review, development, and testing activities along with various testing tools. Following the constraints and standards defined in those guides and leveraging various testing tools allows the identification and removal of numerous vulnerabilities, ultimately contributing to developing a secure application. (Aydos et al., 2022)

Security testing on web applications is primarily concerned with examining business logic, validating inputs, encoding outputs, as well as evaluating authentication and authorization mechanisms. This is carried out to reduce the risk of common vulnerabilities like injection, buffer overflow, file inclusion, and cookie modification (Jaiswal et al., 2014). Security testing involves the usage of automated security tools and manual testing while security professionals' expertise plays a crucial role (Aydos et al., 2022).

The most common approaches used in security testing are black-box and white-box testing (Aydos et al., 2022). White-box testing analyzes the source code and looks for potential vulnerabilities while black-box doesn't have access to the source code, but black-box testing validates system under test (SUT) functionality to perform as expected (Aydos et al., 2022). The different security techniques mentioned in the later section are the variants of these white-box and black-box security techniques.

3.2 Types of Security Analysis Tools in DevSecOps

This section describes the most common categories of automated security analysis tools used within the DevSecOps framework.

3.2.1 Static Application Security Testing (SAST) Tools

Most of the security vulnerabilities arise from the source code. It is necessary to perform security analysis starting from the development process. The only way to prevent security

vulnerabilities and alert developers to their mistakes that could lead to malicious attacks is to make use of some code scan tool (Mateo Tudela et al., 2020).

As mentioned earlier, SAST's purpose is to look for vulnerabilities in a SUT by analyzing its source code. It is a white-box security testing technique that uses different SAST tools to automate the scan and removes the need for a manual review process. SAST tools can be integrated both in IDE and pipelines during software development, which makes it easier to identify and detect the vulnerabilities within the source code. This quicker feedback from the SAST tools is both a cost-effective and time-saving measure. However, the vulnerabilities found by the SAST tools need a final review from the security expert to identify false positives or false negatives as well (Mateo Tudela et al., 2020).

SAST tools can detect OWASP vulnerabilities such as A04:2021-Insecure Design, A03:2021-Injection, A05:2021-Security Misconfiguration, A02:2021-Cryptographic Failures, etc. Some examples of SAST tools are SonarQube (SonarQube, 2024), Snyk (Snyk, 2023), HCL AppScan on Cloud (AppScan, 2024), and so on. (Source code analysis tools - OWASP, 2023)

According to Source code analysis tools - OWASP (2023), some of the strengths of SAST tools are given below:

- It is highly scalable and covers a wide range of programming languages.
- It assists in detecting known vulnerabilities such as Injection, Cryptographic Failures, etc.
- It provides valuable information about the vulnerabilities to the developer by highlighting the problematic code and providing information to resolve it as well.

Additionally, the weaknesses of SAST tools are given below according to Source code analysis tools - OWASP (2023):

- It results in a high number of false positives so manual work from experts is needed to identify it.
- It is limited to detecting only a small portion of application security flaws.
- It might be challenging something to verify the identified vulnerability is truly true positive.

3.2.2 Dynamic Analysis Security Testing (DAST) Tools

Dynamic Analysis Security Testing is black-box security testing that tests a running software application providing malicious payload as input and observes the behavior of that

system to detect the potential vulnerabilities that exist in an application (Mateo Tudela et al., 2020). This can be achieved with the help of DAST tools such as OWASP ZAP (OWASP ZAP, 2024), Acunetix (Acunetix, 2024), Burp Suite (PortSwigger, 2024) etc (OWASP DevSecOps Guideline - V-0.2, 2023).

While performing DAST scans on software under test, the first step is to gain domain knowledge of the software being tested about the application architecture and services. This inspection phase is manually carried out to identify the interface to perform malicious payload attacks. After having sound knowledge of the system, DAST tools are configured to carry out malicious payload attacks on the different interfaces of an application to exploit security flaws. (Mateo Tudela et al., 2020)

On vulnerability scan completion, the report generated by the DAST tools needs to be manually reviewed to filter out the false positive ones, which are fewer compared to those produced by SAST tools. While DAST tools test the running application, it is not possible to integrate these tools into the development process (Mateo Tudela et al., 2020). DAST tools assist in identifying the vulnerabilities listed by OWASP's top 10. The most common ones detected by DAST tools are A03:2021-Injection, Identification and Authorization Failures, and Server Misconfiguration (OWASP DevSecOps Guideline - V-0.2, 2023).

The advantages of using DAST tools are (Setiawan et al., 2020):

- The false positive rate is less compared to SAST tools.
- It is cheaper and flexible to use.

However, a few downsides of DAST tools are (Setiawan et al., 2020):

- It cannot be integrated with the development phase of SDLC, hence doesn't assist in detecting flaws and vulnerabilities from earlier SDLC.
- It doesn't provide valuable insights about the detected vulnerabilities.

3.2.3 Interactive Application Security Testing (IAST) Tools

Interactive Application Security Testing is a security testing method that performs code analysis for detecting vulnerabilities in run-time when there is an interaction on an application. SAST tools run in non-run-time environments while DAST doesn't have access to source code. This testing method combines both SAST and DAST approaches to perform a run-time code base scan. (Mateo Tudela et al., 2020)

The core idea of IAST tools is an agent which is configured into the server side. This agent which acts as a sensor has access to the application source codebase, data and control flow, system configuration, and both client-server components (Mateo Tudela et al., 2020). It monitors and logs the application's interaction behaviors and provides valuable insights about the vulnerabilities in real time. Application interaction can occur either by the end-user or through automated tests. IAST tools introduce fewer false positives and are considered powerful vulnerabilities scanner run time tools (Mateo Tudela et al., 2020). Some examples of IAST tools listed by OWASP DevSecOps Guideline - V-0.2 (2023) are Contrast Community Edition (Contrast Security, 2024), Checkmarx Interactive Application Security Testing (Checkmarx, 2024), Seeker Interactive Application Security Testing (Synopsys, 2024), and HCL AppScan on Cloud (AppScan, 2024).

The benefits of using IAST tools are as follows (Setiawan et al., 2020):

- It helps in the early detection of vulnerabilities in the SDLC process.
- Easy to integrate with CI/CD pipelines
- It serves as a reliable source for detected vulnerabilities as it utilizes the code base of the system.

One drawback of IAST tools is (Mateo Tudela et al., 2020):

- It is highly dependent on language and framework. Therefore, it might not be available to all the programming languages and frameworks.

3.2.4 Software Composition Analysis (SCA) Tools

Software Composition Analysis (SCA) is an automated analysis approach for software to analyze and manage the software's open-source and third-party components. The monitoring is carried out to determine security vulnerabilities and license compliance in third-party libraries (Imtiaz et al., 2021). This will prevent the security risks in a software application such as Supply-Chain attacks that can arise from the third-party component (OWASP DevSecOps Guideline - V-0.2, 2023).

SCA tools should perform this scan frequently before another automated testing such as SAST, and DAST in SDLC to prevent vulnerability results from third-party libraries (OWASP DevSecOps Guideline - V-0.2, 2023). In addition, SCA tools identify the third-party components' vulnerabilities and license compliance based on the known vulnerabilities databases (Imtiaz et al., 2021). OWASP Dependency-check (DependencyCheck,

2024), Snyk (Snyk, 2023), WhiteSource (Mend.io, 2024), and bundler-audit (Bundler-audit, 2024) are some SCA tools (OWASP DevSecOps Guideline - V-0.2, 2023).

Some advantages of using SCA tools are:

- It conducts automatic security assessments for the third-party components.
- It detects the vulnerabilities and provides feedback or automatically eliminates them earlier in the SDLC.
- It assists in managing license-related risks that could arise from open-source components.

3.2.5 Infrastructure Vulnerability Scanning Tools

DevOps combines both development and operation activities. However, security analysis tools such as SAST, DAST, IAST, and SCA are used to scan for vulnerabilities on the development side of the software only. To overcome this limitation, DevSecOps introduces an infrastructure vulnerability scanning approach that looks for vulnerabilities in the network, operation system, and hardware components. Infrastructure vulnerability scanning tools detect the misconfiguration or errors in the infrastructure that hosts the application software. The infrastructure scan can be either authenticated scans or unauthenticated scans. Authenticated scans access the network-based assets using secure protocols or provided credentials. It scans the root level of all the infrastructure services and can detect vulnerabilities. While an unauthenticated scan performs a scan on the publicly available infrastructure. It can result in higher false positive vulnerabilities and is mostly used to determine the security posture of publicly accessible assets. SAINT (Security Administrator's Integrated Network Tool) (SAINT, 2024), Nessus (Tenable, 2024), Arachni (Arachni, 2024), and Burp Suite (PortSwigger, 2024) are some examples of infrastructure vulnerability scanning tools. (OWASP DevSecOps Guideline - V-0.2, 2023)

3.2.6 Container Vulnerability Scanning Tools

The trend of DevOps has introduced the concept of containers where the applications with all their dependencies are developed and bundled in an isolated environment. This makes containers portable and can be deployed easily across different systems. They are popular due to their scalability and efficiency (OWASP DevSecOps Guideline - V-0.2, 2023). However, the security of these containers has become a top concern for DevSecOps or

DevOps engineers to make containerized applications free from vulnerabilities. The container vulnerability scanning tools scan all the packages used in the container and its container image. Like the other security analysis tools, container vulnerability scanning tools list the vulnerabilities present in the container based on the known vulnerabilities list used by the respective container scanning tools (Javed & Toor, 2021). The benefits of using container vulnerability scanning tools are (OWASP DevSecOps Guideline - V-0.2, 2023):

- It helps to identify insecure containers, outdated libraries, and misconfigurations.
- It recommends general guidelines of best practices.

Moreover, the issues with the container security scanner can be (OWASP DevSecOps Guideline - V-0.2, 2023):

- The scan results vary according to the tool used.
- There might be a dilemma with finalizing the configuration settings that a tool can offer.

Clair (Clair, 2024), Anchore (Anchore, 2024), Dagda (Dagda, 2024), Falco (Falco, 2024) and Harbor (Harbor, 2024) are some of the tools that perform container vulnerability scans (OWASP DevSecOps Guideline - V-0.2, 2023).

4 DEVOPS AND DEVSECOPS

The waterfall model, a conventional method of developing software, has already been supplanted in the current software era by other paradigms such as agile/scrum (Koch, 2014), Continuous Integration (Meyer, 2014) and Continuous Delivery (Eberhard, 2014) (CI/CD), Development and Operations (DevOps) (Freeman & Forsgren, 2019), and others. These new paradigms are being introduced to solve the drawbacks of one paradigm vs another with the sole purpose of allowing the delivery of valuable software products flexibly and rapidly. In addition, these new paradigms enable the implementation of security measures in the Software Development Life Cycle (SDLC) from initial stages rather than later (Merkow, 2022).

4.1 DevOps - Development and Operations

The term "cloud" has gained popularity in the software business recently (Alnafessah et al., 2021). Delivering software as a service (SaaS) over the internet as subscription-based has been a novel technique in the cloud. This approach includes various activities, as rapid code integration and continuous delivery to production in collaboration between the development and operation teams is one of them. The cooperation between different teams results in many challenges in the continuous delivery. (Myrbakken & Colomo-Palacios, 2017)

Leite et al. (2019) define DevOps as a “collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions while guaranteeing their correctness and reliability”. As mentioned earlier, DevOps emerged to overcome the complexity that arises while coordinating these various silos team's activities for continuous integration and delivery (Alnafessah et al., 2021).

4.1.1 DevOps Dimension

The term "DevOps" might be perplexing at first since it has several definitions and principles. To achieve a shared understanding of DevOps and its characteristics, Lwakatare et al. (2015) proposed collaboration, automation, measurement, and monitoring as four dimensions. These are the unique characteristics that enable the smooth operation of the continuous integration and deployment cycle of DevOps.

Collaboration: The culture in DevOps involves the collaboration of development and operational teams. This collaboration results in information, knowledge sharing, and exchanging feedback about the product, which makes sure that all the team members are on the same page.

Automation: It automates all the tedious activities superseding manual work. It includes activities such as automating tests and automating infrastructure configuration with Infrastructure as Code (IaC). Automation is carried out throughout the SDLC process to be compliant with Agile software development and make continuous code integration and delivery possible.

Measurement: It aids in gaining valuable information and feedback about the system for improvement from insights gained not only through testing but also from system logs in the production environment.

Monitoring: It can continuously monitor the different logs, expectations, bugs, and system status in the production environment. As a result, it can proactively or post actively alert about failure or malicious activities. This also makes it possible to gather the feedback for the next continuous cycle of the DevOps.

These dimensions of the DevOps are the foundation for the DevOps lifecycle.

4.1.2 DevOps Cycle

DevOps extends the agile methodologies which aim to unify the overall development and operational activities into a sequential set of processes (Demiliani et. al, 2020). Figure 1 illustrates the various continual stages of the DevOps lifecycle for build and release.

The stages that are on the left-hand side in Figure 1 depict the development period, while the stages on the right represent the operating period, which in combination brings both the development and operation teams under the same roof for the development of the product. The cycle involves six stages which are carried out iteratively together with continuous feedback to achieve an iterative continuous release until the product goal is aligned with customer requirements (Yarlagadda & Teja, 2021).

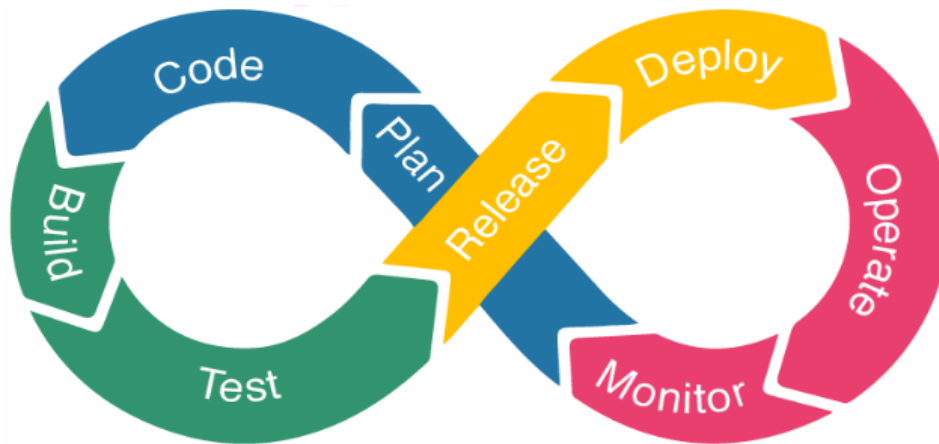


Figure 1. Stages of DevOps Cycle, adapted from (Yarlagadda & Teja, 2021)

Each stage is described as follows.

Plan: It involves gathering requirements and defining the scope and objectives for software product development (Alnafessah et al., 2021). Afterward, the product backlog is created according to requirements and scope, which will be broken down into iterative smaller deliverables aims to achieve a product goal (Gokarna, 2020).

Code and Build: It involves mostly developers, who turn the defined plan into the working version of software artifacts with development activities using a certain programming language(s) and the series of code integration. At the end of this stage, software artifacts are made available for testing. (Alnafessah et al., 2021)

Test: This stage involves mostly automated testing activities to ensure the quality of the developed software artifacts (Gokarna, 2020).

Release and Deploy: It begins with the release stage, where it verifies the product/software artifacts built from previous stages both in terms of requirement and release readiness. Subsequently, the verified software product is continuously deployed to the production environment via automated pipelines. (Alnafessah et al., 2021)

Operate: This stage includes the activities to make sure that the product runs smoothly after it is deployed in the production environment. It is more about configuration for achieving high reliability, scalability, and flexibility. (Alnafessah et al., 2021)

Monitor: The final phase of the “DevOps lifecycle” focuses more on the analysis of customer behavior on the application usage along with the environment on which it runs. It

mostly collects logs and analyzes them to provide feedback to improve the software in the next iteration of the DevOps lifecycle (Alnafessah et al., 2021).

4.1.3 DevOps Pipeline and Tools

Figure 2 represents different phases that are designed to follow the principles defined in DevOps dimensions (Beetz & Simon, 2022). However, implementing these phases in practice involves continuous integration and continuous delivery (CI/CD).

Continuous integration (CI) ensures that new changes are build, tested, and merged in the shared repository through an automated process, while continuous delivery (CD) is an automated process of deploying changes to production. (Gokarna, 2020)

DevOps CI/CD pipelines are a vital part of the DevOps workflow which simply automates building, integrating, testing, and deploying processes making quality delivery of software (Gokarna, 2020). The succession of the DevOps approach and its pipeline relies on having the right DevOps tools in place to support them. DevOps tools are software application that enables automation and orchestration of the DevOps pipelines. Each stage in the pipeline utilizes different DevOps tools to perform specific tasks. (Gokarna, 2020)

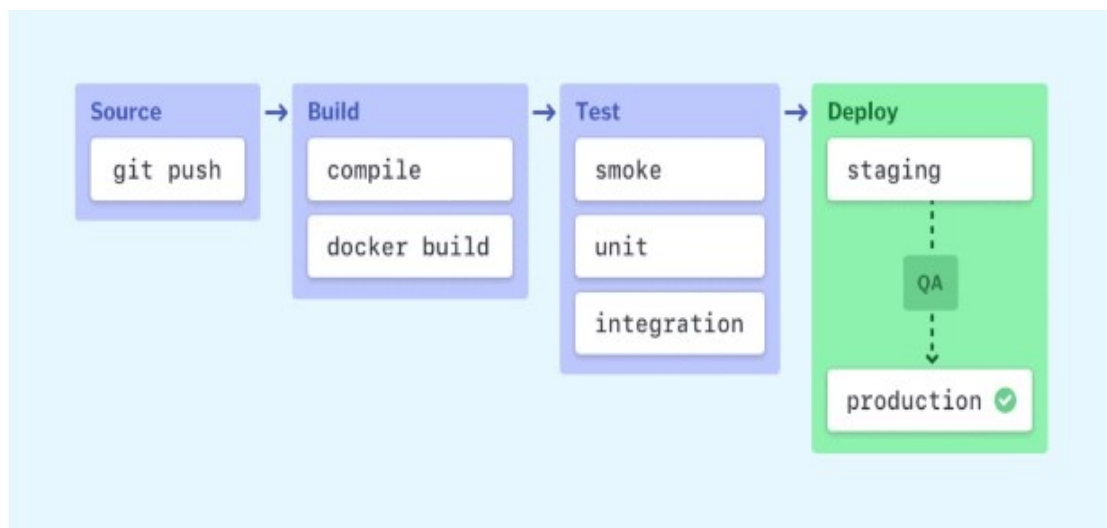


Figure 2. Stages of DevOps CI/CD pipeline (Anastasov, 2022)

Figure 2 represents the different basic stages involved in the DevOps CI/CD pipeline. They are:

Source: It is the initial stage that triggers the CI/CD pipeline when new changes are pushed (Anastasov, 2022). In this stage version control tools such as Git is used to track and manage the code changes (Gokarna, 2020).

Build: During this stage, the code is compiled to generate an artifact suitable for deployment. Building an artifact is carried out either through a language-specific build process or within a Docker container. In addition, this stage tracks down a compilation error present in a code. If any errors are present, then it terminates the pipeline progress to another stage and results in failure. (Anastasov, 2022) Either continuous integration DevOps tools such as Jenkins (Jenkins, 2024), Travis CI (Travis-CI, 2024), Microsoft Build (Build, 2024) etc., or containerization tools like Docker (Docker, 2024) are used in this stage for building an artifact (Gokarna, 2020).

Test: This stage involves testing the artifacts from the build stages with automated unit, smoke, and integration tests. This stage acts as a safeguard to prevent the integration of a faulty code into the production environment. It verifies that the code functions correctly on its own and its integration does not negatively impact the code overall. If this stage fails, then immediate attention is needed from the developer to fix the issue. (Anastasov, 2022) Junit (JUnit5, 2024), Jasmine (Jasmine, 2024), and Selenium (Selenium, 2024) are some examples of DevOps tools that are used in this stage for continuous and automated testing (Gokarna, 2020).

Deploy: When both build and test stages are passed, then it is ready to be deployed. However, there might be various deployment environments such as staging or QA for the internal team and production for the end users. The deployment strategies might vary according to the different organizations as the most common one is from staging to QA to production (Anastasov, 2022). DevOps tools such as Ansible (Ansible, 2024), Chef (Progress"Chef", 2024), Docker (Docker, 2024), Puppet (Puppet, 2024) etc. are used to deploy the tested code to production in an automated manner (Gokarna, 2020).

Besides the CI/CD pipeline stages of the DevOps, various other DevOps tools incorporate the remaining stages of the DevOps cycle. For instance, tools like JIRA (Atlassian, 2024) and Trello (Atlassian, 2024) are used for continuous planning, while Grafana (GrafanaLabs, 2024) and Prometheus (Prometheus, 2024) a tools that assist in continuous monitoring. Additionally, Slack (Slack, 2024), and Open Web Analytics (Open Web Analytics, 2024) are used for gathering continuous feedback (Gokarna, 2020).

Consequently, DevOps pipelines integrate different tools through different stages to ensure smooth, automated, and continuous integration and delivery. The ultimate goal of DevOps pipelines and tools is to support DevOps practices. The basic stages of the DevOps pipelines are given in Figure 2 however, stages and usage of tools can vary according to the requirements of an organization. DevOps pipelines are the guidelines that integrate different tools to automate the software development process and promote collaboration between development and operation teams. Furthermore, it provides reliable and high-quality software delivery.

4.1.4 Benefits and Shortcomings

DevOps significantly improves the collaboration and knowledge sharing between the development and operations teams (Krief, 2022). Every team member's role becomes transparent under DevOps. Furthermore, working in short iterations with the customer allows for early feedback and keeps the application bug-free from the start (Senapathi et al., 2018). Subsequently, it minimizes the time to market with continuous integration, deployment, and improvement and assists to be always aligned with the business goal (Krief, 2022).

DevOps also poses several challenges. Firstly, introducing the DevOps paradigm can lead to a drastic transformation in workplace culture, potentially causing reluctance from the team to this change in culture. In addition, a lack of skilled manpower in a team with expertise in both development and operations could hinder the successful integration of DevOps in an organization. Moreover, the transformation of the monolithic application to a cloud-based and microservices application to enable DevOps is considered a complex, challenging, and time-consuming task because all the required tools for continuous integration and deployment need to be set up, all while dealing with finding or training the skilled staff to manage the new technology stack. Furthermore, different teams work in collaboration which might sometimes confuse the role of each member in this DevOps paradigm. (Senapathi et al., 2018)

On the other hand, the DevOps paradigm often omits implementing security activities due to the concern that it could slow down the release process and many organizations have been avoiding doing so. For instance, Figure 2 also does not incorporate security testing activities within the DevOps pipeline. Security has always been a crucial concern in

software development which most of us overlooked while integrating the DevOps approach (Myrbakken & Colomo-Palacios, 2017). DevSecOps – Development, Security, and Operations is the new paradigm that tries to address security needs in development and operation activities by extending the DevOps principles which we will discuss more in the next upcoming section 4.2.

4.2 DevSecOps – Development, Security and Operation

As previously stated, Development, Security, and Operation (DevSecOps) is introduced to overcome the shortcomings of security activities in DevOps. DevSecOps extends DevOps workflow along with its principles and emphasizes on shift-left approach to integrate security activities from the earlier stages in SDLC (Rajapakse et al., 2021).

DevSecOps approach is crucial for assessing the security of an application while doing continuous integration and deployment because all the potential security issues are identified earlier before release. In addition, DevSecOps aims to enhance the quality of the software delivery. (Lombardi & Fanton, 2023)

4.2.1 DevSecOps Principles

Myrbakken & Colomo-Palacios (2017) extend DevOps principles with culture, sharing, and shifting security to the left to define key principles of DevSecOps. The following paragraph highlights the key principles of DevSecOps.

In the DevSecOps culture, development, operation, and security teams along with the customer collaborate closely for integrating security activities from the beginning in SDLC (Sánchez-Gordón & Colomo-Palacios, 2020). This helps to flatten the responsibilities of security as well as the operation and development among all the stakeholders. In addition, security requirements are always considered either when implementing new features or responding to feedback (Myrbakken & Colomo-Palacios, 2017). Furthermore, it emphasizes automating the security testing as well at different stages in DevOps (Myrbakken & Colomo-Palacios, 2017). Security knowledge is required among the team members for using those security tools integrated into different phases of SDLC. Therefore, it encourages sharing security knowledge with the team

members to use those security tools and continuously develop an application with security in mind (Sánchez-Gordón & Colomo-Palacios, 2020). DevSecOps introduces security activities starting from the earlier phases of SDLC introducing shift left culture. Security testing can be started from the non-functional requirement itself and move towards the functional requirement which assists the developer in tracing security issues as early as possible. This removes the traditional waterfall approach towards security testing and is a more cost-effective measure for security testing. (Anjaria & Kulkarni, 2022)

4.2.2 DevOps Vs DevSecOps Pipeline

DevOps pipelines are meant to automate the software delivery process and different DevOps tools are essential for executing tasks like building, testing, deploying, and operating (Gokarna, 2020). Figure 2 illustrates the different general stages and associated activities within the DevOps pipeline. One noticeable gap in Figure 2 is the absence of security activities in the DevOps pipeline.

DevSecOps extends the DevOps pipeline to introduce a shift-left approach by integrating different security activities in the pipeline starting from the non-functional requirements. However, there are no standard guidelines for integrating security tools in the pipelines, as it varies from company to company according to the requirements (Lombardi & Fanton, 2023).

Table 2 represents the comparison between the DevOps and DevSecOps general pipelines stages and additional security activities conducted on the DevSecOps paradigm are highlighted with a bold text.

Table 2. DevOps pipeline stages vs DevSecOps pipeline stages

Stage	DevOps	DevSecOps
Source	Coding is done to implement the features. The new changes are pushed to the central git repository. (Gokarna, 2020)	Secure coding is done with the help of linting and secret-checking tools . As a result, secure code is pushed to the git repository. (Marandi et al., 2023)
Build	Code is compiled to create a working version of artifacts (Gokarna, 2020).	Static Application Security Testing (SAST) tools are used before building code to track down different flaws or vulnerabilities (Marandi et al., 2023).
Test	Automated unit testing, integration testing, and smoke testing are performed to make sure that everything works as intended without any errors (Gokarna, 2020).	In addition to DevOps basic testing procedures, Dynamic Application Security Testing (DAST) is carried out to simulate the application attacks to identify vulnerabilities if exist (Marandi et al., 2023).
Deploy	Build artifacts are deployed to production (Gokarna, 2020).	Same as in DevOps but deployed with an additional security layer before the application known as a Web Application Firewall (WAF) (Marandi et al., 2023).

Furthermore, Figure 3 depicts the standard DevOps and DevSecOps pipeline activities with gray boxes and purple boxes respectively. DevSecOps pipeline introduces vital security activities alongside the existing DevOps pipelines (Lombardi & Fanton, 2023).

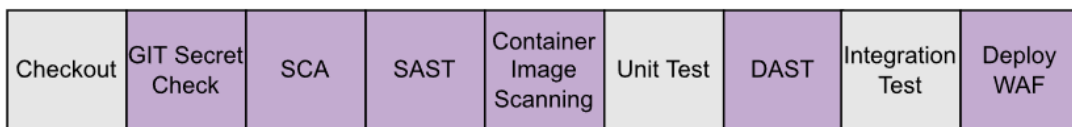


Figure 3. The gray square represents DevOps pipelines while purple indicates additional security activities in the DevSecOps pipeline, (Lombardi & Fanton, 2023)

The principal security testing categories that are introduced in the Figure 3 pipeline to conduct security testing in pipelines are Secret check, Software composition analysis (SCA), Static application security testing (SAST), Container image scanning, Dynamic Application Security Testing (DAST), and Web application firewalls (WAF). A secret check is a process of scanning code repositories in search of sensitive information such as passwords, keys, tokens, or any secret that has been pushed or committed (Lombardi & Fanton, 2023). SCA is a process that helps to identify known vulnerabilities and license compliances in third-party packages or libraries (Lombardi & Fanton, 2023). SAST is a process that performs automatic scanning of the code to identify any security vulnerabilities or known issues (Lombardi & Fanton, 2023). This assists in detecting security vulnerabilities earlier in the development process and assists in providing immediate feedback or security alerts to the developer (Lombardi & Fanton, 2023). Container image

scanning scans the content and metadata of the container images within the CI/CD pipelines for any security vulnerabilities or misconfiguration (Lombardi & Fanton, 2023). DAST is a run-time application security testing run before production deployment to simulate the action of an attacker or user to discover any vulnerabilities in the application (Lombardi & Fanton, 2023). WAF helps to monitor or filter incoming and outgoing traffic between a web application and the internet. It protects an application from a common attack, such as SQL injection, and so on (Lombardi & Fanton, 2023).

4.2.3 DevSecOps Tools

The above security activities introduced by Lombardi & Fanton (2023) are further mapped to the DevSecOps pipelines stage and tools that are associated with them in Table 3. Most of the tools mapped with the security activities are inspired by the Azure DevSecOps guidelines and best practices (Azure, 2023), except for some which I chose to be used for comparison in the later chapter.

Table 3. DevSecOps stages and its security activities and tools

Stages	Security Activities	DevSecOps Tools	Tool Type
Source (Commit)	Git Secret Check	GitHub secret scanning (Azure, 2023)	Open Source
		CredScan (Azure, 2023)	Proprietary
	Software Composition Analysis (SCA)	OWASP Dependency Check (Azure, 2023)	Open Source
		Snyk (Snyk, 2023)	Proprietary
		WhiteSource (Katalinic et al., 2021)	Proprietary
		Dependabot (Azure, 2023)	Open Source
	Static Application Security Testing (SAST)	SonarCloud (Azure, 2023)	Open Source
		Snyk (Snyk, 2023)	Proprietary
		Checkmarx (Katalinic et al., 2021)	Proprietary
Build and Test	Dynamic Application Security Testing (DAST)	OWASP Zed Attack Proxy (ZAP) (Azure, 2023)	Open Source
	Infrastructure Scanning	Tfsec (Azure, 2023)	Open Source
		Checkov (Checkov, 2023)	Open Source
		Snyk (Snyk, 2023)	Proprietary
	Container Scanning	Microsoft Defender (Azure, 2023)	Proprietary
Deploy	WAF	Azure WAF (Azure, 2023)	Proprietary

4.2.4 Mapping DevSecOps Security Testing Activities with OWASP Top 10 Vulnerabilities

Table 4 highlights possible OWASP Top 10 vulnerabilities that a security activity can detect in the DevSecOps lifecycle. Table 4 is mapped based on the identification of OWASP's top 10 vulnerabilities by various security tools, as reported in various research papers. Rahman & Williams (2021) have correlated CWE with vulnerabilities. We further identified the corresponding OWASP Top 10 vulnerabilities according to the OWASP mapped CWE with OWASP vulnerabilities and security activities are determined with the DevSecOps guidelines (OWASP DevSecOps Guideline - V-0.2, 2023). Besides that, Elder et al. (2022), Goncalves (2022), Li (2020), and Srokosz et al. (2018) have respectively analyzed various security tools and mapped their finding with the OWASP Top 10 vulnerabilities. Therefore, we presented all the collected information from various studies in Table 4.

Table 4. Correlation between DevSecOps security activities and the OWASP Top 10 vulnerabilities

DevSecOps Security Activities	OWASP Top 10 Vulnerabilities (2021)
Git Secret Check	Identification and Authentication Failures (Rahman & Williams, 2021).
Software Composition Analysis (SCA)	A02:2021-Cryptographic Failures, A03:2021-Injection, A06:2021-Vulnerable and Outdated Components, A01:2021-Broken Access Control, A08:2021-Software and Data Integrity Failures (Gonçalves, 2022).
Static Application Security Testing (SAST)	A02:2021-Cryptographic Failures, A05:2021-Security Misconfiguration, A01:2021-Broken Access Control, A03:2021-Injection (Li, 2020). A08:2021-Software and Data Integrity Failures, A04:2021-Insecure Design, A07:2021-Identification and Authentication Failures (Elder et al., 2022).
Dynamic Application Security Testing (DAST)	A08:2021-Software and Data Integrity failures, A02:2021-Cryptographic Failures, A05:2021-Security Misconfiguration, A03:2021-Injection, A01:2021-Broken Access Control, A07:2021-Identification and Authentication Failures, A04:2021-Insecure Design (Elder et al., 2022).
Infrastructure Scanning	A01:2021-Broken Access Control, A05:2021-Security Misconfiguration, A08:2021-Software and Data Integrity Failures, A07:2021-Identification and Authentication Failures, A02:2021-Cryptographic Failures. (Rahman & Williams, 2021) (OWASP, 2021)
Container Scanning	A05:2021-Security Misconfiguration, A03:2021-Injection, A06:2021-Vulnerable and Outdated Components (OWASP DevSecOps Guideline - V-0.2, 2023).
WAF	A03:2021-Injection, A10:2021-Server-Side Request Forgery (SSRF) (Srokosz et al., 2018).

4.2.5 Benefits and Challenges

In the DevSecOps approach, security is always in the state of mind of every stakeholder. This makes it possible to consider security throughout the SDLC starting from planning, developing, deploying to operating. In addition, automation testing shifts the waterfall approach of testing towards various stages of SDLC which assists in tracing code smells or vulnerabilities earlier and makes cost effectiveness in cost saving. Furthermore, continuous monitoring of the logs provides valuable insight into different security incidents that occur in an application. Moreover, DevSecOps improves the security aspects of software delivery. (Myrbakken & Colomo-Palacios, 2017)

However, it has several challenges that exist in different aspects such as tools, practices, infrastructure, and people. Firstly, there is no standard procedure for selecting different security tools as the effectiveness of DevSecOps success highly depends on these tools. In addition, it is not possible to fully automate the security activities to replace the manual security testing approach. Further, it is quite difficult to integrate DevSecOps principles in a complex cloud environment. Finally, all the stakeholders are not aware of the security, and collaboration for knowledge sharing about security between the stakeholders might be quite challenging as there is a culture shift. (Rajapakse et al., 2022)

4.3 Related Works

In recent years, DevSecOps has been a buzzword for researchers and numerous different studies have been carried out for adopting DevSecOps principles and defining a unified framework to integrate various security tools. In addition, various categories of automated security tools have been integrated into the DevSecOps pipeline and the overall effectiveness of the DevSecOps pipeline. These recent studies on DevSecOps provide a differential perspective on DevSecOps and its practices.

Aljohani & Alqahtani (2023) proposed the DevSecOps unified framework to address the challenges in establishing a secure software application delivery process. The two SAST tools, Dependency Check, which emphasizes Software Composition Analysis (SCA), and Checkov, static analysis tools designed for Infrastructure as Code (IaC) are introduced in the pipelines along with Nikto Scanner as a DAST tool. The effectiveness

of these security tools in the purposed DevSecOps pipelines is observed by running it against the Damn Vulnerable Java Application.

Bizjak (2023) conducted research to explore the practical implementation of DevSecOps pipelines, with a particular focus on the lower execution time of the CI/CD pipeline. SonarQube, OWASP Dependency-Check, and Gitleaks are selected as the static security analysis tools to perform SAST and SCA scans, while OWASP Zap was introduced as a DAST tool. The execution time of the pipelines was less than 10 minutes while executing the SAST and DAST test on the WebGoat, a vulnerable application through the DevSecOps Azure pipeline.

Putra & Kabetta (2022) studied the implementation of DevSecOps pipelines. They covered all the lifecycle stages of DevOps and introduced SAST using Njsscan and DAST using OWASP-ZAP in the continuous testing stage. In the end, the dockerized application was deployed to production using the DevSecOps approach through GitLab.

Rangnau et al. (2022) integrated three various DAST tools in the DevSecOps pipelines and evaluated them in terms of DevSecOps principles. The integrated tools were integrated in CI/CD pipelines and automated dynamic security testing was performed against the known vulnerable application named OWASP WebGoat. They concluded that integrating DAST tools is quite challenging in terms of achieving speed and setting it up on the CI/CD pipeline. Expertise is required for setting up manual tests to automate the DAST tools. Furthermore, they highlighted the trend of using a variety of DAST tools in the CI/CD pipeline to achieve wide coverage to detect vulnerabilities.

Viitasuo (2020) conducted studies to add various security activities to the DevOps approach using open-source security tools. SonarQube as SAST tool, Clair as container vulnerability scanning tool, OWASP dependency check as SCA tool, and OWASP ZAP as DAST tool are different tools selected to run the Damn Vulnerable Web Application in a CI/CD pipelines. SAST, SCA, and container scanning tools are easy to integrate compared to DAST tools which need more effort to set up. The author points out poor documentation as a major setback for using open-source security tools. In addition, the majority of vulnerabilities listed in the OWASP Top 10 were successfully detected by the use of open-source tools in DevSecOps pipelines.

Ibrahim et al. (2022) focused their studies on securing Infrastructure as Code (IaC) with the DevSecOps approach. Tfsec was introduced as an IaC tool to monitor the security code smells in the terraform file. The author primarily focused on evaluating the effectiveness of Tfsec with the DevSecOps approach and found it efficient in terms of detecting misconfiguration and execution speed.

In summary, various research has been made on DevSecOps principles, tools, and the framework for executing the DevSecOps CI/CD pipeline efficiently over the recent years. We found that the area of DevSecOps is vast and still evolving. According to our study, recent research is mostly focused on the SAST and DAST security activities. Moreover, the selection of the right security tools has been marked as one of the most challenging tasks in developing the DevSecOps pipeline and it was reported that most of the organizations have ended up integrating two or more same-category security test tools to achieve wider coverage in detecting vulnerabilities. At last, table 5 summarizes the recent works' key takeaways in a tabular format.

Table 5. Related work key points comparison

Studies	SAST tools	DAST tools	SCA tools	IaC tools	Execution Time	CI/CD platform	Benchmark / Test environment
(Aljohani & Alqahtani, 2023)	-	Nikto Scanner	Dependency Check	Checkov	-	Shuffle	Damn Vulnerable Java Application
(Bizjak, 2023)	SonarQube, GitLeaks	OWASP ZAP	OWASP Dependency Check	-	Yes	Azure DevOps	WebGoat
(Putra & Kabetta, 2022)	Njsscan	OWASP ZAP	-	-	Yes	Gitlab	Not specified
(Rangnau et al., 2022)	-	OWASP-Zap, Selenium, Base, JMeter,	-	-	Yes	Gitlab	WebGoat
(Viitasuo, 2020)	SonarQube, Clair	OWASP Zap	OWASP Dependency Check	-	-	GoCD	Damn Vulnerable Web Application
(Ibrahim et al., 2022)	-	-	-	Tfsec	-	Amazon	Own

5 METRICS OF ANALYSIS AND STUDY DESIGN

This section introduces the tool selection criterion, evaluation metrics, and test environment used to evaluate both SAST and IaC tools for this study. In addition, the study design includes six different Azure CI/CD pipelines for performing automated vulnerability scans against two vulnerable goat applications in the Azure environment. Afterward, manual inspection was conducted on each tool vulnerabilities scan's output and unified them in two different CSV files for SAST and IaC tools. Based on this CSV file dataset, the different metrics were evaluated.

This study design and metrics of analysis concern only the SAST and IaC security tools. They are considered the first and most proactive approach for security testing and continuously deploying secure modern web applications and their infrastructure in the Azure cloud. They are seamlessly easy to integrate into the Azure pipeline and can be the starting point for any organization that wants to make a secure deployment with the DevSecOps paradigm. In previous works, only a few IaC tools have been evaluated. Therefore, to fill the gap and provide insight into the starting tools for the organization that wants to shift towards the DevSecOps approach, we selected both SAST and IaC in our study.

5.1 Tools Selection Criterion and Evaluation Metrics

This section introduces the SAST and IaC security tools selection and evaluation metrics.

5.1.1 Tools Selection Criterion

The focus of these studies is to evaluate the different SAST and IaC tools. This limits our boundary for selecting the best 3 SAST and IaC tools respectively. The main criteria for selecting these automated tools are as follows:

- a. The tools must be either open source or free to use with some limitations.
- b. The tools should have good documentation and communities.
- c. The tools must be able to integrate into the Azure CI/CD pipelines.
- d. The results generated by the tools should be presented within the Azure pipeline itself or should easily integrate with the web dashboard of respective tools for a clear display of results.

5.1.2 Tool Evaluation Metrics

The two various sets of evaluation metrics are defined for SAST and IaC respectively due to the use of two different benchmarks. The SAST tools focused on finding the OWASP vulnerabilities whereas IaC tries to detect the security smells in infrastructure code. Afterward, the findings from these selected tools are evaluated based on the metrics respectively defined below.

Firstly, performance metrics are simply the set of metrics that define how well the security tools perform in terms of their efficiency and effectiveness in identifying security issues (Marandi et al., 2023). The metrics for evaluating SAST tools' performance are described below:

- True Positive (TP): It indicates the count of a genuine vulnerability presence in the source code which is correctly detected by the tools (Kuszczyński & Walkowski, 2023). In this study, the true vulnerabilities were identified on the scan results by cross-referencing the predefined ground truth of each benchmark. In addition, for the rest of the vulnerabilities detected by the tools that are not listed on the ground truths, we performed either the code inspection, manual review, or manual penetration testing, to determine true cases.
- False Positive (FP): It highlights the total vulnerabilities found by a tool that is not present in the source code (Kuszczyński & Walkowski, 2023). Code inspection and manual review were conducted for the detected vulnerabilities by the tools and flagged as FP for vulnerabilities which are irrelevant ones, which can't be exploited, and false alarms that do not possess any security risks on its exploitation. Additionally, an in-depth understating of the vulnerabilities was also needed to flag them as FP.
- True Negative (TN): It refers to a count of false vulnerabilities in the source code that were not reported by the tool. False vulnerability simply refers to the vulnerability detection by a tool that does not pose a genuine security risk. (Kuszczyński & Walkowski, 2023) This metric was quite challenging to calculate as the benchmark has not specified any true negative cases. Hence, we create a list of all the unique false vulnerabilities that are detected by all tools. Then, we count the number of false vulnerabilities that a tool was able to ignore to flag as vulnerabilities. This gives us a True Negative count for a tool.

- False Negative (FN): It keeps the count of the inability to identify the vulnerabilities that exist in the source code (Kuszczyński & Walkowski, 2023). It was calculated with the help of the ground truth that was not able to be detected by the tool.
- Total: It is the overall vulnerabilities identified by a tool that remain the same across all tools within an application (Kuszczyński & Walkowski, 2023).

$$Total = TP + FP + TN + FN$$

- Precision: It helps to determine how accurate positive predictions are. Greater precision numbers signify a higher degree of accuracy in identifying actual vulnerabilities. (Higuera et al., 2020)

$$Precision = \frac{TP}{(TP + FP)}$$

- Recall: It is a True Positive Rate, which is the proportion of truly identified vulnerabilities to the overall count of actual vulnerabilities present in the source code (Higuera et al., 2020).

$$Recall = \frac{TP}{(TP + FN)}$$

- F_β -Score: It is an F-measure metric also known as a weighted harmonic mean of recall and precision. It helps to balance the precision and recall with the coefficient beta where $\beta < 1$ emphasizes its focus on precision while $\beta > 1$ puts more attention on recall. (Higuera et al., 2020)

$$F - measure = \frac{(2 * Precision * Recall)}{(Precision + Recall)}$$

$$F_\beta - score = (1 + \beta^2) * \frac{(Precision * Recall)}{((\beta^2 + Precision) + Recall)}$$

- Accuracy: It is a measure that indicates the percentage of the vulnerabilities that are correctly reported and correctly unreported, out of all vulnerabilities reported by the tool (Kuszczyński & Walkowski, 2023).

$$Accuracy = \frac{(TP + TN)}{Total}$$

F_β-Score is one of the key metrics in this study that is used to conclude the tools because as weighted harmonic means of recall and precision, this score can be used to emphasize either recall ($\beta > 1$) when minimizing false negatives is important or precision ($\beta < 1$) when minimizing false positives are priorities. Furthermore, the F-measure balances both precision and recall when ($\beta = 0$). Accuracy is our other key interest metric that indicates how effectively a tool can predict the vulnerabilities correctly.

In addition to the above metrics, we add additional metrics for analyzing the SAST tools. This additional metric highlights the general capabilities of a static analysis tool which are important while selecting them for DevSecOps pipelines.

- CI/CD Integration support: It lists the possible cloud platform for CI/CD integration.
- License: It evaluates the associated licensing aspects.
- Output format: It assesses the output format that makes it possible to integrate the generated report in the generic CI/CD pipeline.
- Language support: All the possible programming languages that are possible to scan with the selected SAST tool.
- Tool Type: Basically, determines whether the tool is Open-Source Software (OSS) or Commercial.
- Limitation: It highlights the possible drawbacks of using the SAST tool if any.
- Azure DevOps Extension Support: This Azure DevOps extension support provides the tool's build task from the Visual Studio marketplace which you can specify to it YAML configuration file while developing the pipeline.

Secondly, we didn't find any metrics for evaluating IaC tools in the past works. Terragoat contain known security smells and are classified according to their severity.

Consequently, we concluded to evaluate the IaC tools according to the above-mentioned metrics used for SAST tools evaluation.

5.2 Selected Tools

The selection of both SAST and IaC tools is based on the criterion defined in section 5.1.1. In our study, we considered cost and flexibility to integrate into the Azure pipeline as the most important aspects for selecting the tools.

SonarQube is considered the most popular choice of SAST tool for performing SAST based on the past works defined in section 4.3. SonarCloud is preferred over SonarQube in this study due to its seamless integration into the CI/CD pipelines whereas SonarQube is the self-managed solution that needs to be installed in the server.

Besides that, Snyk is an exception in our tool selection case. We chose Snyk to observe how well commercial products perform against the open-source tools with Snyk providing both SAST and IaC scans. Tfsec is chosen as it is mentioned in the previous work described in section 4.3 and meets the selection criteria. However, Checkov is the other IaC tool that is designed to detect all the vulnerable by-design security smells present in Terragoat and it also meets our selection criteria. Lastly, Semgrep is also a SAST tool available as open source and can be implemented either into CI/CD pipelines or local editor to perform code scans. The basic details of all the selected tools are given below respectively, allowing us to understand their nature and role in Azure CI/CD pipelines.

5.2.1 SonarCloud

SonarCloud is a SAST tool that is used to perform Static Application Security Testing (SAST) for a broad range of 26 programming languages. It has a predefined set of rules that are used to evaluate against our code base to detect vulnerabilities or defects. It can be easily integrated into the pipelines and can be configured as per the need. Effective usage of SAST tools can be made during a pull request creation, where they can address the issues encountered in the development process. (SonarCloud, 2023)

SonarCloud performs static code analysis as mentioned earlier to detect coding mistakes, bugs, and security vulnerabilities. It assists in detecting issues early which enhances the

quality of the code. It doesn't need an application to run for performing analysis. It can be easily integrated with various Git repositories such as GitHub, Bitbucket, Azure DevOps, and Gitlab. SonarCloud offers free scans only for the public repository. (SonarCloud, 2023)

5.2.2 Semgrep

Semgrep is a lightweight, open-source static analysis tool that helps to find bugs and dependency vulnerabilities in the source code. It enforces code standards with over 2000 existing rules across 30 different languages. It can be run either in CI or local code editor or on the command line. This makes it easier to deploy, manage, and monitor Semgrep at scale as well. (Semgrep, 2024) Semgrep OSS Engine - an open-source engine to perform scans, Semgrep Cloud Platform (SCP) - dashboard for managing all the scans and reports which can be continuously integrated into different cloud providers such as GitHub, GitLab, and more, Semgrep Code - scan source code with Semgrep's Pro rules and Semgrep's Pro Engine, Semgrep Supply Chain (SSC) - scanner to detect vulnerabilities in open-source third-party libraries, Semgrep Secrets - secrets scanner to detect unintentional secrets in the source codebase are different products from Semgrep ecosystem (Semgrep, 2024).

5.2.3 Snyk

Snyk is a commercial tool that can be used for both SAST and IaC analysis. It also offers a free plan with restricted features. Software Composition Analysis (SCA), Container Security or Container Vulnerability Scans, and Cloud Security are the additional scans that can be performed by Snyk. (Snyk, 2023)

Snyk code is responsible for performing source code analysis whereas, Snyk IaC conducts the static analysis of IaC codes. Snyk relies on its proprietary database, known as Snyk Intel Vulnerability Database. This database is maintained through the efforts of a dedicated research team, developer community, and proprietary research with evolving security threats. (Snyk, 2023)

5.2.4 Tfsec

It is a security scanner available as an open source for static analysis specially designed for scanning Infrastructure as Code (IaC) written in terraform only. It is designed to work

in CI/CD pipelines as well as locally. In addition, the output of the tfsec scan is user-friendly, and comprehensive documentation of the vulnerabilities makes it possible to resolve the detected vulnerabilities rapidly and easily. (tfsec, 2023)

Static analysis and seamless integration with the official HashiCorp Configuration Language (HCL) parser are the tfsec's developer-first strategies for scanning IaC templates. This approach makes the security vulnerabilities detection earlier and fixed before any infrastructure changes come into effect. (tfsec, 2023)

5.2.5 Checkov

It is also an open-source tool for performing static analysis over Infrastructure as Code (IaC). Security smells that could potentially lead to vulnerabilities are detected earlier by Checkov according to its 750 predefined policies. It scans Infrastructure as Code (IaC) across various platforms such as Helm, Azure Resource Manager, Kubernetes, Terraform, Serverless framework, Docker and CloudFormation. In addition, custom policies can be defined to detect any cloud resource misconfiguration that is not defined in Checkov's predefined policies. It is maintained by the global community. (Checkov, 2023)

The summary of the selected tools with their version, category, and tool type is given in Table 6.

Table 6. Summary of the selected SAST and IaC tools

S.N.	Tool Name	Version	Category	Tool Type
1	SonarCloud	4.8.0.2856	SAST	Commercial
2	Snyk Code	1.1267.0	SAST	Commercial
2	Semgrep	1.55.1	SAST	OSS
3	Tfsec	1.26.0	IaC	OSS
4	Checkov	3.1.46	IaC	OSS
5	Snyk IaC	1.1267.0	IaC	Commercial

5.3 Study Design Specification

Firstly, benchmarks are simply the set of test cases that assess the accuracy, coverage, and speed of security tools that detect vulnerabilities. The well-known benchmarks used for evaluating security tools for web applications are the Web Application Vulnerability Scanner Evaluation Project (WAVSEP), Open Web Application Security Project (OWASP), and Web Input Vector Extractor Teaser (WIVET) (Mburano & Si, 2018). However, only one benchmark project named Terragoat, which specializes in evaluating IaC vulnerability scanner tools was found.

Secondly, this section describes two of the benchmarks OWASP benchmark and Terragoat, which are cloned in our Azure DevOps repository. Our study will focus on evaluating the SAST and IaC tools in the Azure CI/CD pipeline. The research approach we used for the evaluation of those automated security tools is to run these tools against these benchmarks in the Azure test environment, collect the scan data, and evaluate their performance based on the tool evaluation metrics defined in the earlier section. Overall, this section outlines the study design used to assess both SAST and IaC tools.

5.3.1 Selected Benchmarks

This section covers in detail descriptions of the selected benchmarks for both SAST and IaC tools.

OWASP Juice Shop, developed by Bjoern Kimminich, is supported by a volunteer team and operates under the MIT License as an open-source project (OWASP Juice Shop, 2023). We chose the OWASP Juice Shop as our benchmark from the OWASP collection to analyze our SAST tools. This choice is made because it is a vulnerable web application constructed entirely in JavaScript, reflecting the prevalent trend in many real-world web applications (OWASP Juice Shop, 2023). OWASP Juice Shop is known as a modern JavaScript web application that simulates real-world web security vulnerabilities. This application is built using the latest backend and frontend technologies in web development like Node.js, Express, and Angular. It is favored for training, testing, and research purposes as it covers the latest OWASP Top 10 web vulnerabilities. (OWASP Juice Shop, 2023)

This application can be used in numerous different ways. One can study the different vulnerabilities that can present in the modern angular web application, or one can try out different hacking challenges on the OWASP Juice shop to uncover the existing vulnerabilities in the vulnerable application. On the other hand, one can use this as a “guinea pig” application to validate the performance of different security vulnerability scanner tools against modern front-end technologies and REST APIs. (OWASP Juice Shop, 2023)

The OWASP Juice Shop’s version 15.0.0 is used in our test environment. The vulnerable code snippet was presented on various hacking challenges on its code base. We collected that list and considered it as our ground truth for the SAST tools evaluation. Appendix A represents the 32 different ground truths present in OWASP Juice Shop.

Terragoat is a vulnerable-by-design benchmark repository for the IaC terraform code (Bridgecrewio, 2023). It is designed for educational and training purposes to demonstrate how typical security smells can inadvertently pushed into the production cloud environment. This benchmark project provides valuable insights for DevSecOps engineers in detecting and testing security smells on infrastructure as code. (Bridgecrewio, 2023)

This project is available on both Gitlab and Github and offered under the MIT License. As mentioned earlier, this project empowers the DevSecOps pipeline in creating and implementing a robust strategy to prevent security smells on IaC code. It also serves as a tool for testing policy-as-code frameworks such as Bridgecrew and Checkov, inline linters, pre-commit hooks, or alternative code scanning techniques. This project is offered in multiple cloud environments like AWS, Azure, and GCP and follows the path of a traditional goat project which serves as guidelines for implementing secure development practices in cloud infrastructure development. (Bridgecrewio, 2023)

Terragoat’s version 0.6.0 is used in our environment, and it serves as a guinea pig for various cloud infrastructures such as Alicloud, AWS, Azure, GCP, and Oracle. We focused only on Azure and collected about 137 known security smells across for it which will serve as ground truth for our IaC tools evaluations. Appendix B represents all the ground truth along with its relevant check ID, number of occurrences, and severity.

5.3.2 Test Environment

This section introduces the key technologies and approaches used in setting up the benchmark projects and the security tools to automate the scans on those setup projects. The entire test environment is created in the Azure environment using the student tier subscriptions (Microsoft Azure for Students, 2024). Figure 4 represents the bird's eye view of the test environment, which consists of the Azure Cloud Environment and Azure DevOps.

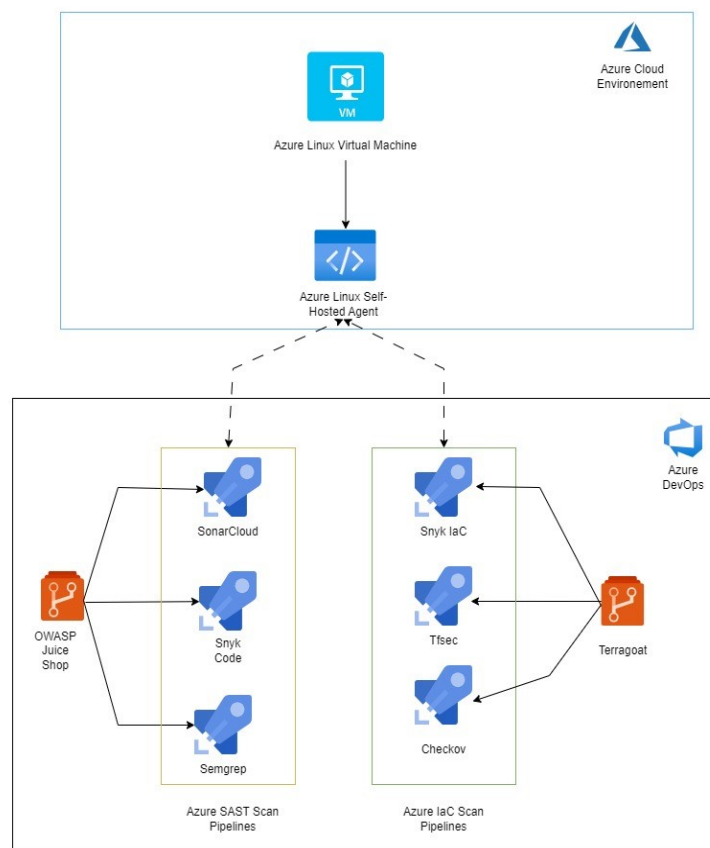


Figure 4. High-level configuration of the test environment

Figure 4 architecture consists of the following components.

Azure Cloud Infrastructure

In this work, we focused on running six independent pipelines to demonstrate the DevSecOps pipelines for various SAST and IaC scans in the Azure cloud environment. For this purpose, we created a self-hosted Linux build agent (Azure Pipelines, 2024) to build and perform different scans on two different repositories. We used one Azure virtual

machine (VM) on Azure Cloud to host the self-hosted Linux build agent which runs as a service on it. Azure self-hosted agent is configured with Azure DevOps to run all the pipelines in an organization. For Azure VM, we selected it to be Linux with an image of Ubuntu Server 20.04 LTS – x64 Gen2, North Europe as a region, and keeping the rest of the field default. The VM was of size B1ms with 1 vCPUs, 2 GiB RAM, 2 data disks, 640 max IOPS, and 4 GiB local storage.

Azure DevOps Configuration

Azure DevOps is a comprehensive service that consists of all the tools in DevOps such as Kanban board, pipeline, repository, test plans and artifacts in one place. This service is offered by Microsoft. In one word, Azure DevOps Services provides a collaborative environment for developers, project managers, and contributors to develop, test, and deploy the software product efficiently (Microsoft, 2023).

We used the student tier account to create an Azure DevOps which is enough to execute different automated scan pipelines for our studies. Afterward, we cloned the two above benchmarks from their public repository and pushed their source code into the two separate repositories on our Azure DevOps organization. We created two service connections in Azure DevOps for performing Snyk and SonarCloud scans. This was the initial setup done on the Azure environment on which the various SAST and IaC scans are performed.

Azure CI/CD Pipeline

We have selected three SAST tools and three IaC tools as shown in Figure 4. Consequently, six independent Azure pipelines have been configured in the Azure DevOps, each set to trigger manually. Each of the two benchmark repositories has three pipelines to perform respective SAST and IaC scans based on the tool types. The self-hosted agent on Linux is set up on the Azure cloud environment to run these pipeline jobs. After the completion of each scan, they produce their vulnerability scan reports which are analyzed to conclude those tools. The following describes the respective pipeline configurations done to achieve those SAST and IaC scans.

SonarCloud pipeline: The SonarCloud task from the Azure DevOps pipeline extensions is required to develop SAST SonarCloud scan Azure pipelines. A service connection must be configured with the SonarCloud token, which can be set up from the service connection menu in the project settings of Azure DevOps. A SonarCloud account is necessary

to acquire the token. Additionally, Java 17 or above needs to be configured on build agents to run this pipeline. After the scans are finished, the results can be viewed on the SonarCloud dashboard. It's important to note that this scan applies only to public repositories. Appendix C.1 represents the yml configuration file for the SonarCloud Azure pipeline.

Semgrep pipeline: As there was no task available from Azure, pipeline scripts need to be configured manually. In addition, the UsePythonVersion@0 task was not working as expected, so a manual download of the Python on the build agent directory was necessary to make the scans possible. Similarly, we create a Semgrep account to get an app token to link the results from scans to the Semgrep dashboard. Likewise, the repository name, branch, and URL on which the scans are performed are saved as variables that are needed in this pipeline. Appendix C.2 illustrates the configuration file (YML) for the Sempgrep SAST Azure pipeline.

Snyk pipeline: It also doesn't have any Azure extension that will ease writing Azure pipelines for Snyk scans. The Snyk access token needs to be saved as a variable to perform the scans, which can be obtained by creating an account on Snyk. Appendix C.3 illustrates the pipeline configuration for performing SAST code scan using Snyk. The output of the scans was printed on the pipeline itself.

Checkov pipeline: It is used for IaC as it doesn't have any Azure build-in task, manually scripts are written on the pipelines. It is dependent on Python to perform scans, so manual configuration of the Python was needed on the build agent to make scans running possible. The output of the scan results was published as test results using PublishTestResults@2 Azure tasks. Azure pipeline configuration for Checkov is shown in Appendix C.4.

Snyk IaC pipeline: Most of the configurations are similar to the Synk configuration illustrated in Figure 8 besides the command to execute the IaC scans. Appendix C.5 shows the YAML configuration file for performing Snyk IaC scans.

Tfsec pipeline: Tfsec tasks are available as an Azure extension from the marketplace, which can be used in the pipelines directly. This is the easiest configuration among all the pipelines mentioned above. Appendix C.6 demonstrates the configuration of the Azure pipeline for tfsec.

5.3.3 Data Collection

Each tool has its way of representing its scanned results on Azure pipelines. Therefore, we carried out the manual walkthrough of the scan results to collect and unify them in a uniform form manner in CSV files. On the one hand, columns named SAST tool, file-name, line number, OWASP Category, CWE, Severity, False Positive, True Positive, found by Snyk, found by SonarCloud, and found by Semgrep were made to collect the SAST tools scan results in the tabular format. Similarly, found in Terragoat, recommendation, checkov id, severity, terraform file and resource name, found in checkov, found in tfsec, tfsec severity, tfsec id, tfsec description, found in Snyk, Snyk severity, Snyk id, Snyk description, found in Checkov, Checkov severity, Checkov id and Checkov description was column named created to contain the IaC tools scan results in another CSV file.

As a summary of this section, these pipelines are triggered manually in a sequential manner from the Azure DevOps. In addition, these pipelines represent two different parts of security activities i.e., SAST and IaC vulnerability scans in DevSecOps. Likewise, scan results are separately analyzed to evaluate the metrics to determine the effectiveness of different tools. These security activities are the static analysis which can be run with the IDE or in pre-commit hooks as well.

6 RESULTS AND DISCUSSION

This research utilized three distinct SAST tools and three different IaC tools for analyzing their behavior and performance against OWASP Juice Shop and Terragoat, respectively. This section emphasized and discussed the various metrics derived from the scan conducted by each tool, outlining both the strengths and limitations of individual tools.

6.1 SAST Tools

The motive of this section is to examine the SASTs tools based on the metrics derived from the outcomes of their execution against the OWASP Juice Shop. Firstly, metrics like True Positive, False Positive, False Negative, False Positive rates, precision, accuracy, recall, and F-score have been shown based on the vulnerabilities detection from each tool in Figure 6 and Figure 7. On the other hand, we gathered the general metrics about the SAST tool such as Language support, CI/CD Integration support, License, Tool Type, Output format, and Limitation which is illustrated in Table 7, and drew a general summary for each tool based on the metrics.

6.1.1 Vulnerability Detection and Metrics

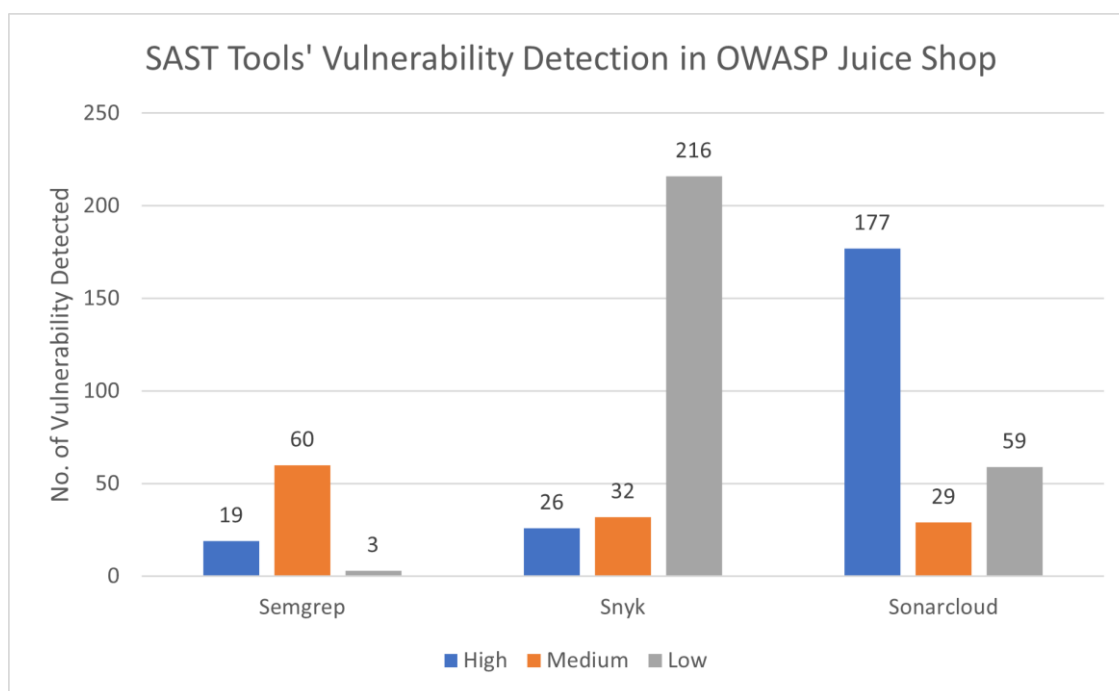


Figure 5. SAST tools' vulnerability detection in OWASP Juice shop

Figure 5 represents the vulnerabilities detected by each tool in terms of its severity. At first glance, we got that Snyk had the higher detection rate while Semgrep had the lower

compared to others. However, we cannot determine much from this detection alone. Therefore, we further classified these detections into True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN).

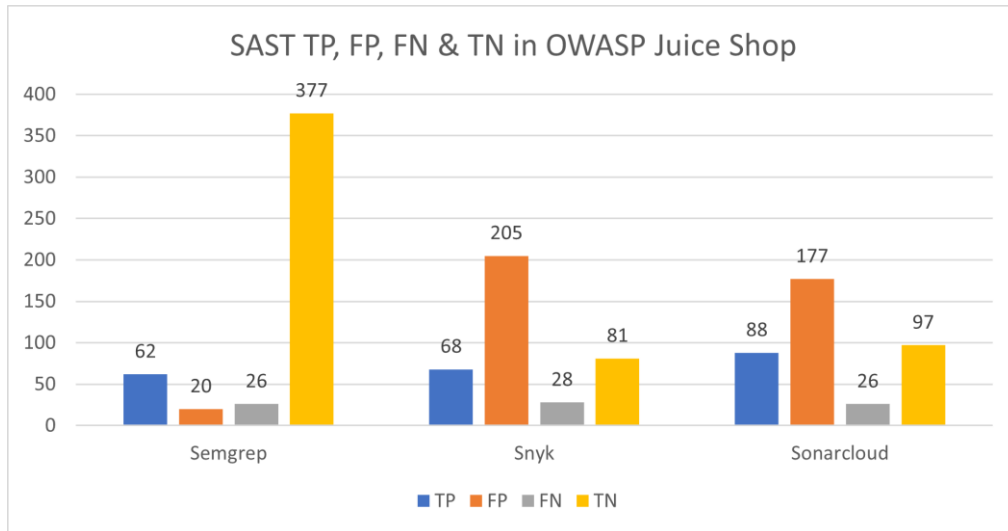


Figure 6. TP, FP, FN, and TN detection in OWASP Juice Shop

Figure 6 offers important insights regarding the performance of each tool. These insights include:

- While Semgrep has a relatively high True Negative (TN) count indicating truly ignoring false vulnerabilities detection and has the less False Negative (FN) among the other tools, meaning fewer missed known vulnerabilities. In addition, it has a relatively low number of False Positive and False Negative rates, indicating that it can perform well in identifying both true and false vulnerabilities.
- Snyk has a higher number of True Positive (TP) count indicating good at detecting vulnerabilities. However, it has a higher False Positive (FP) detection, indicating a significant number of incorrect predictions. These high FP counts lead to a manual investigation for identifying false alarms, which is simply a waste of time and resources.
- SonarCloud has the highest True Positive (TP) count among the tools, indicating strong vulnerability detection. We considered including security hotspots in SonarCloud as vulnerabilities, which led to a notable increase in the rate of False Positives (FP).

We further applied these metrics to calculate precision, recall, accuracy and F1 score, providing a more thorough assessment of each tool's performance which can be illustrated by Figure 7 given below.

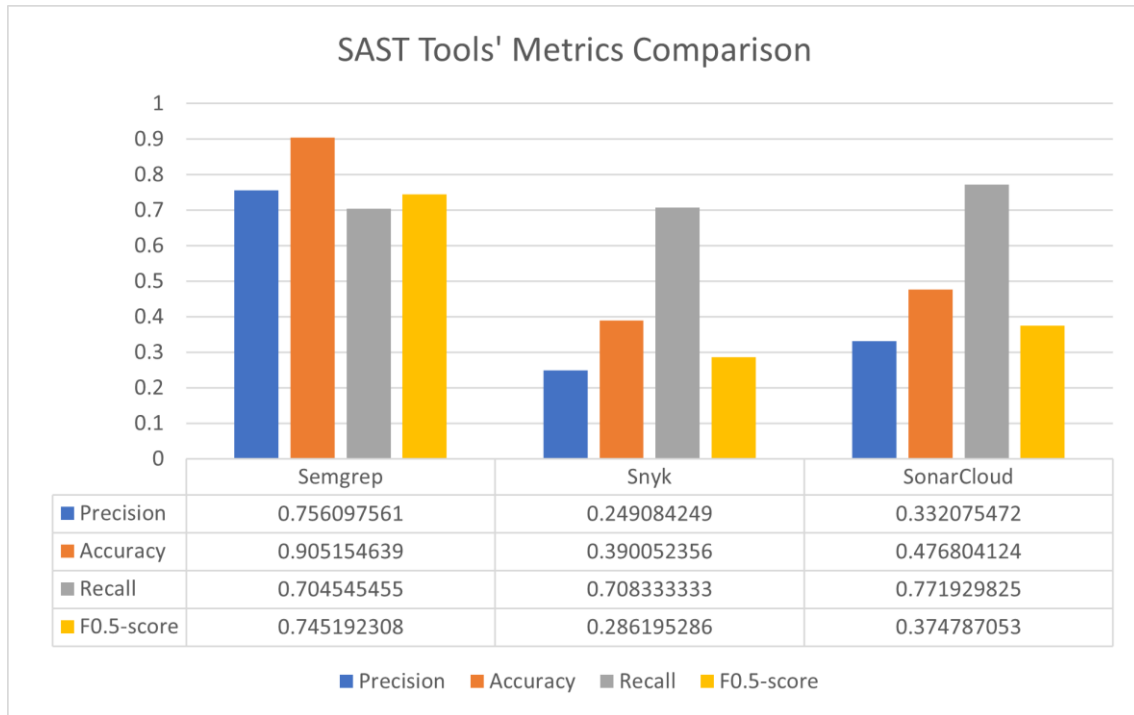


Figure 7. SAST tools' metrics comparison

The outcomes of each tool from Figure 7 based on metrics are analyzed and discussed below:

- **Precision:** It represents the proportion of true positives to the total of true positives and false positives, with higher precision indicating a reduced rate of false positives and a tool with a higher precision value is likely to produce correct predictions i.e., true positive (TP) (Higuera et al., 2020). Semgrep has a higher precision of 0.756, meaning about 75.6% of vulnerabilities detected are true positive whereas Snyk and SonarCloud have a lower precision value of 0.249 and 0.332 respectively. Snyk has the lowest true positive rate i.e., 33.2%.
- **Accuracy:** It is the ratio of accurate prediction (TP and TN) relative to the overall count of vulnerabilities detected (Higuera et al., 2020). Semgrep has the highest accuracy at 0.905 i.e., 90.5% on identifying the presence or absence of vulnerabilities. In contrast, Snyk exhibits the lowest accuracy at 0.390, while SonarCloud produces an overall correct prediction of about 0.476 i.e., 47.6%.

- **Recall:** The true positive rate which represents the proportion of true positive results to the count of real positive cases (Higuera et al., 2020). SonarCloud has the highest recall of 0.772, indicating that it has a lower rate of false negatives and identifies the highest actual vulnerabilities at about 0.772 i.e., 77.2%. On the other hand, Semgrep and Snyk have recall values of about 0.704 and 0.708 respectively. As a result, Semgrep became the tool with the lowest rate of identifying the actual vulnerabilities.
- **F_{0.5}-score:** It's essentially the weighted average of recall and precision (Higuera et al., 2020). We focused on determining the fewer false positives so kept the β as 0.5. Semgrep has the highest F-measure score i.e., 0.745, showing a balanced performance between precision and recall. Leaning slightly towards precision. It indicates its effectiveness in minimizing the false positives. However, Snyk has the lowest F-score value with 0.286 which aligns with its precision. On the other hand, SonarCloud reflects slightly better precision with 0.374 than Snyk.

6.1.2 Summary

Table 7 provides general benchmarking information about various SAST tools. This benchmarking consists of general information about the tools such its version, type, license, and code base language it supports for scans. All this general information is collected through its official website. Furthermore, it provides information about specific Azure DevOps CI/CD extensions along with other different types of CI/CD pipelines it supports for integration. The possible output format for displaying results with the time it takes to execute their SAST scans is also provided. Moreover, the limitations that were found while performing these tools' scans are also noted in Table 7. The speed of execution is collected by running those pipelines on our Azure test environment.

Table 7. *General benchmarking metrics for SASTs Tools*

Metrics	Semgrep	Snyk Code	Sonar Cloud
Version	1.55.1	1.1267.0	4.8.0.2856
Language support	PHP, TypeScript, Kotlin, Scala, Rush, JSX, Java Python, C#, JSON, Terraform, Generic, JavaScript, Ruby, GO.	Kotlin, Ruby, C#, Swift, JavaScript, PHP, Scala, Go, TypeScript, VB.NET, Java, C/C++.	TypeScript, Docker, Ruby, Terraform, Swift, ABAP, CSS, HTML, PL/SQL, Scala, C++, CloudFormation, VB.NET, Python, JavaScript, Kubernetes, Apex, C#, Objective-C, Java, COBOL, XML, RPF, Flex, T-SQL, GO, VB6, ARM.
CI/CD integration support	GitHub, GitLab, Bitbucket, and Azure DevOps	GitHub, Bitbucket Cloud, Azure DevOps, Gitlab, AWS, GCP	GitHub, Bitbucket, Azure DevOps, Gitlab, CircleCI, TravisCI, Other
License	LGPL 2.1	Not OSS	Not OSS
Tool type	OSS	Commercial	Commercial
Output format	Emacs, JSON, GitLab SAST, GitLab Secrets, JUnit XML, SARIF, Vim formats	JSON, SARIF, HTML	SonarCloud dashboard
Limitation	Semgrep Cloud Platform is free for up to 10 contributors	100 scans per month	Free scans only on a public repository
Speed of Azure pipeline scans	3 minutes 1 second	1 minute 31 seconds	1 minute 57 seconds
Azure DevOps extensions support	No	No	Yes

Based on the information gathered according to general metrics in Table 7, it is evident that SonarCloud had the most language support among the three SAST tools. Similarly, Semgrep had a greater number of output format support compared to others. Mainly, it has the JUnit support for results which will make it easier to publish as a test artifact and analyze results in the Azure pipelines itself. Snyk has less flexibility towards output format, while SonarCloud directly publishes the scan results on their portal. In addition, SonarCloud experienced the shortest execution time within the Azure pipeline for conducting SAST scans. Furthermore, it also had the Azure extension support, simplifying its integration into the Azure pipeline in comparison to other tools. However, each tool presented its own set of limitations. Although Semgrep proves to be user-friendly for smaller teams, challenges arise with larger teams, necessitating the need to use the paid version for utilizing the Semgrep Cloud Platform to facilitate easier scanned results analysis. Besides this, SonarCloud doesn't offer free usage for private projects and the free version of Snyk limits the number of scans per month.

We concluded the presence of the OWASP vulnerability category in the tool by inspecting its detection of true positive vulnerabilities, including its associated CWE, and extending CWE's mapping with the OWASP's vulnerability category. Additionally, the ground truths of OWASP Juice Shop helped to know the presence of different OWASP Vulnerabilities in OWASP Juice Shop itself. The subsequent Table 8 displays OWASP vulnerability categories present in the OWASP Juice Shop, along with indications of whether the SAST tools successfully detected them (yes or no). It shows that Semgrep is the one that detects most of the OWASP vulnerabilities, while the vulnerability named A09:2021–Security Logging and Monitoring Failures is not detected by these SAST tools.

Table 8. OWASP categories detection by the SAST Tools

OWASP Vulnerability Categories	Present in OWASP Juice Shop	Semgrep	Snyk Code	SonarCloud
A01:2021-Broken Access Control	Yes	Yes	Yes	Yes
A02:2021–Cryptographic Failures	Yes	Yes	Yes	Yes
A03:2021-Injection	Yes	Yes	Yes	Yes
A04:2021-Insecure Design	Yes	Yes	No	No
A05:2021–Security Misconfiguration	Yes	Yes	Yes	Yes
A06:2021-Vulnerable and Outdated Components	No	Yes	No	No
A07:2021–Identification and Authentication Failures	Yes	Yes	Yes	Yes
A08:2021–Software and Data Integrity Failures	Yes	Yes	Yes	Yes
A09:2021–Security Logging and Monitoring Failures	Yes	No	No	No
A10:2021-Server-Side Request Forgery (SSRF)	No	Yes	Yes	Yes

In summary, Semgrep is the most accurate in correctly identifying vulnerabilities with fewer false positives due to its good $F_{0.5}$ -score and accuracy. Additionally, Semgrep is an open-source tool and can be seamlessly integrated into Azure pipelines despite not having any Azure DevOps extension for CI/CD integration. Although Semgrep took more time to execute pipelines, its integration into its Semgrep Cloud Platform provides more flexibility while analyzing results. Secondly, Snyk has a greater number of false positives which produced poor performance across various metrics due to low precision, accuracy, and $F_{0.5}$ -score. Despite having a reasonable recall, indicating its ability to identify a significant amount of actual vulnerabilities with a cost of higher false positive rates. Furthermore, Snyk has a limitation of 100 scans per month with very limited support of output formats for displaying results. It has the fastest scan capabilities among the SAST tools. Lastly, SonarCloud achieved the highest recall value, indicating its effectiveness in identifying actual vulnerabilities. However, its lower precision values and accuracy of less than 50% make it vulnerable to detecting higher false positive value results with significant incorrect predictions. Sonar Cloud's free version is limited to scanning for public repositories only and has a moderate scan execution time compared to the rest of the tools. Moreover, it facilitates scans for a diverse array of programming languages.

6.2 IaC Tools

Similarly, this section aims to evaluate IaC tools by analyzing their performance against the Terragoat and present the findings in Table 10 and Figures 8, 9, 10, and 11. Firstly, sub-section 6.2.1 includes Figure 11 which displays metrics such as precision, accuracy, recall, and F-score, accompanied by values for true positive, false positive, false negative, and false positive rate. These data were derived from the vulnerabilities detected by the IaC tools. Secondly, Table 9 provides the mapping of common vulnerabilities found by all IaC tools. Conversely, the consecutive sub-section 6.2.2 provides Table 10, which presents general metrics for the IaC tool, including Supported IaC types, cloud providers support, CI/CD integration-specific support, license, tool type, output format and limitations. In addition, an overall summary of each tool is provided.

6.2.1 Vulnerability Detection and Metrics

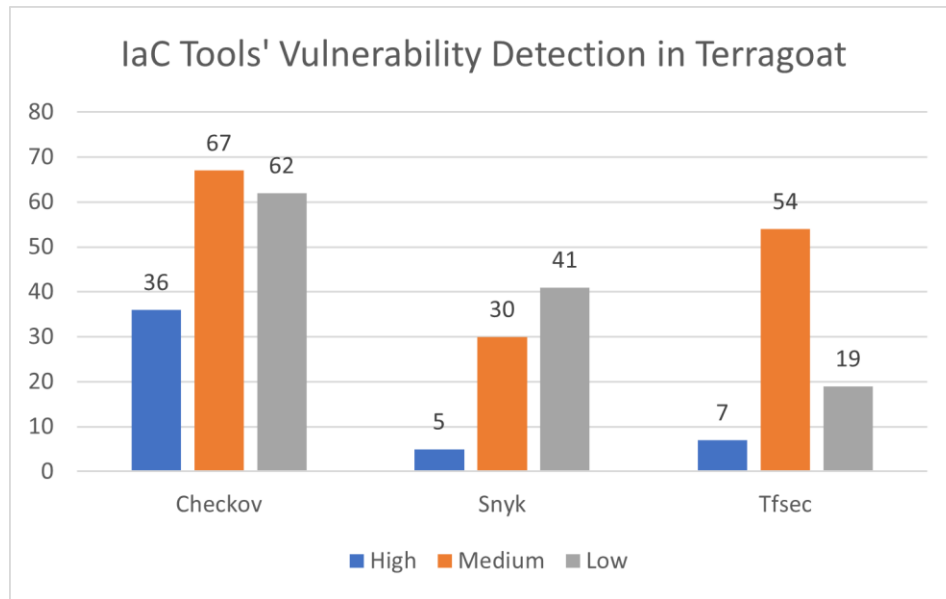


Figure 8. IaC tools' vulnerability detection in Terragoat

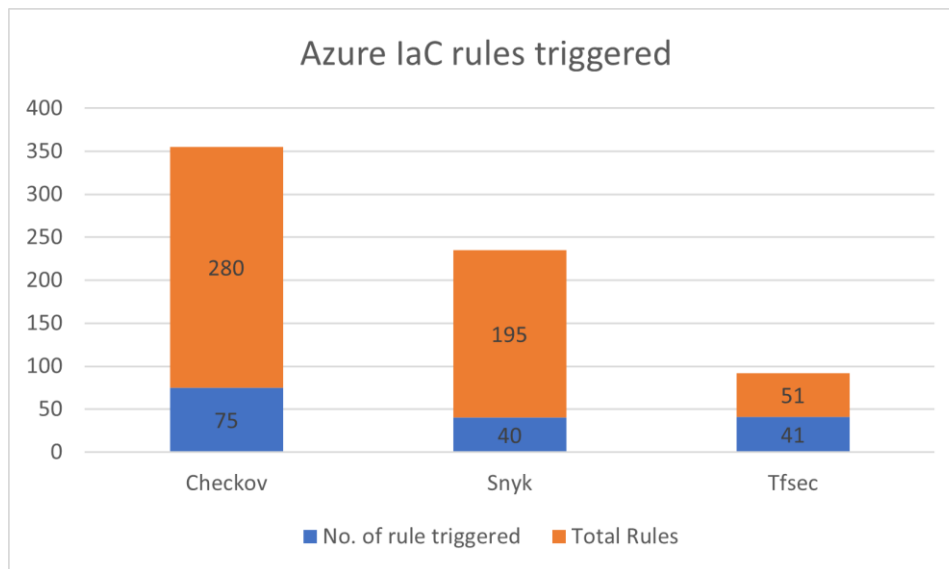


Figure 9. Azure IaC rules triggered by each IaC tool

Figure 8 illustrates the severity-based distribution of vulnerabilities detected by each tool which are the initial results found after performing scans while Figure 9 represents the total Azure cloud policies or rules available in comparison to the ones that were triggered to detect vulnerabilities.

Initially, Checkov has the higher detection rate due to its larger number of rules coupled with a higher rate of rule triggers which is illustrated in Figure 9. Moreover, Tfsec has the

second highest rate of detection with most of their available rules triggered to find vulnerabilities. Finally, Snyk has the lowest rate of vulnerability detection in terms of other tools although it had a higher set of Azure IaC policies. Nevertheless, we further compared IaC tools to its TP and FN rates where FP and TN remain 0 for all tools in our experiment against Terragoat.

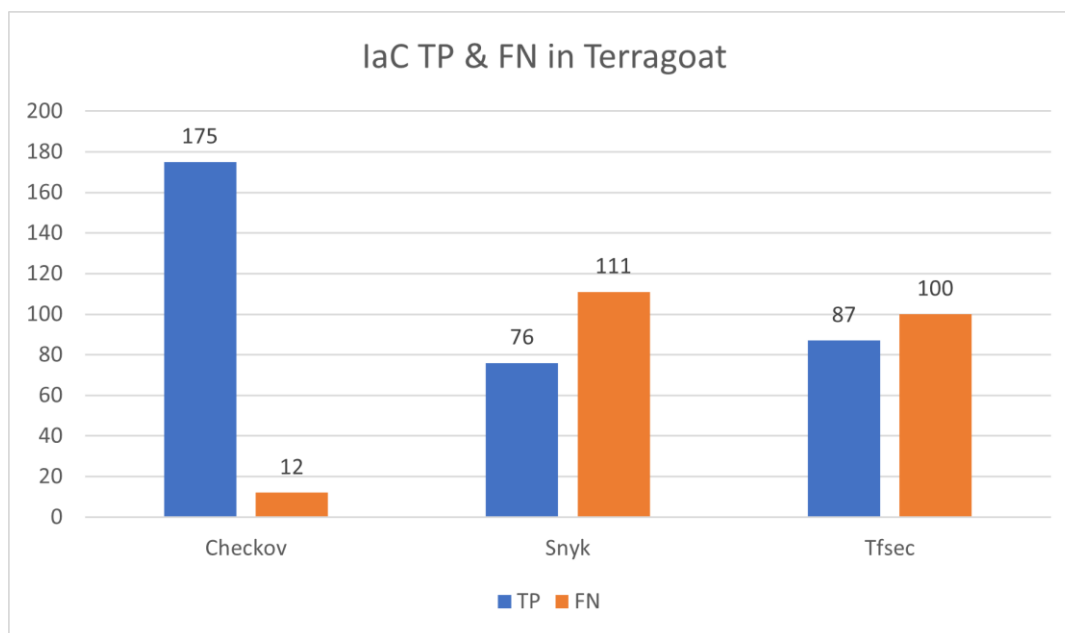


Figure 10. True Positive and False Negative obtained by IaC tools

Figure 10 explains the rate of both True Positive and False Negative, while True Negative and False Positive remain 0 for all the IaC tools. The following insights can be obtained from the Figure 10:

- Checkov demonstrates superior performance by correctly identifying the highest number of issues with minimal false negatives. This suggests that Checkov accurately identifies a significant portion of vulnerabilities in Terragoat, with a minimal occurrence of failing to detect true vulnerabilities.
- Snyk has a greater number of missed vulnerabilities detection (FN) than its actual vulnerabilities detection (TP). Out of the 137 known vulnerabilities on Terragoat, it detected the lowest number of true positives, specifically 76, which is lower compared to other tools. In addition, it is also accompanied by a higher count of false negatives among the other tools.

- Tfsec at least performed better than Snyk in terms of vulnerability detection which successfully identified 87 true positives and failed to be detected 100 vulnerabilities.

However, it's essential to emphasize that Terragoat was developed by the same team that also created Checkov. Consequently, there might be a bias favoring Checkov in this comparison. This represents a limitation in our IaC assessment.

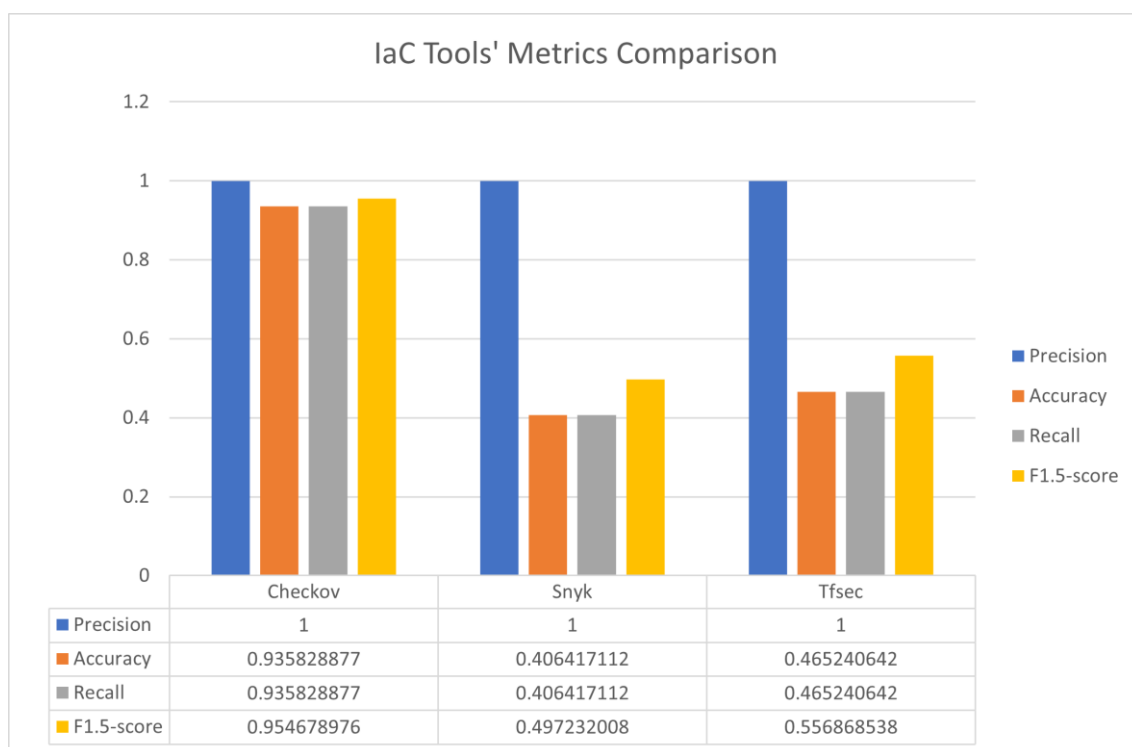


Figure 11. IaC tool's metrics comparison

The results of each tool from Figure 11 are analyzed and discussed based on the metrics below:

- Precision: It represents the proportion of correctly identified true positives out of all true and false positives cases. Since the false positive is 0 for all the tools, we cannot analyze much from this as the value of precision is the same for all tools.
- Accuracy: Checkov has the highest accuracy at 0.935 i.e., 93.5% on identifying the presence or absence of vulnerabilities. In contrast, Tfsec produces an accurate prediction of about 0.465 i.e., 46.5%, while Snyk has the lowest accuracy at 0.406.
- Recall: Checkov exhibits the highest recall of 0.935, indicating that it has a lower rate of false negatives and identifies the highest actual vulnerabilities. On the other

hand, Snyk and Tfsec have recall values of about 0.406 and 0.465 respectively. As a result, Snyk became the tool with the lowest rate of identifying the actual vulnerabilities.

- $F_{1.5}$ -score: We focused on minimizing false negatives so kept the value of β as 1.5. Checkov has the highest F-measure score i.e., 0.954, showing a balanced performance between precision and recall. Leaning slightly towards recall. It indicates its effectiveness in minimizing false negatives. However, Snyk has the lowest F-score value with 0.497. On the other hand, Tfsec reflects slightly better precision with 0.556 than Snyk.

6.2.2 Mapping of the Matched Vulnerabilities in Tool Comparison

The following Table 9 represents the mapping of vulnerability issues found in Terragoat with unique IDs of Checkov, Tfsec, and Snyk which are Checkov Check Id, Tfsec Id, and Snyk Id respectively. These vulnerability issues are found by all these tools on their respective scans. Out of 78 known unique issues in Terragoat, 34 were the most common that both tools were able to detect.

Table 9. Mapping of found vulnerabilities in Terragoat with Checkov Id, Tfsec Id, and Snyk Id, extension of (Bridgcrewio, 2023)

SN	Issue	Severity	Checkov Check Id	Tfsec Id	Snyk Id
1	SSH access does not restrict public access from the internet.	Critical	CKV_AZURE_10	AVD-AZU-0047 AVD-AZU-0050	SNYK-CC-TF-33
2	RDP access is possible by anyone from the internet.	Critical	CKV_AZURE_9	AVD-AZU-0048 AVD-AZU-0047	SNYK-CC-TF-33
3	The expiration date is not specified for all keys.	Critical	CKV_AZURE_40	AVD-AZU-0014	SNYK-CC-TF-173
4	Not all secrets have an expiration date.	Critical	CKV_AZURE_41	AVD-AZU-0017	SNYK-CC-TF-174
5	The auditing setting is not activated for SQL servers.	High	CKV_AZURE_23	AVD-AZU-0027	SNYK-CC-TF-167
6	RBAC remains disabled on AKS clusters.	High	CKV_AZURE_5	AVD-AZU-0042	SNYK-CC-TF-80
7	Basic authentication is utilized by Azure Instance (Instead consider using SSH Key).	High	CKV_AZURE_1 CKV_AZURE_178	AVD-AZU-0039	SNYK-CC-TF-263
8	'All' is not selected for the 'Threat Detection types' setting.	High	CKV_AZURE_25	AVD-AZU-0028	SNYK-CC-TF-168
9	Alert notifications are not set up for MSSQL servers.	High	CKV_AZURE_26	AVD-AZU-0018	SNYK-CC-TF-169
10	Azure-managed disk's encryption is disabled.	High	CKV_AZURE_2	AVD-AZU-0038	SNYK-CC-TF-77
11	Azure Monitoring configuration is not set up for AKS logging.	Medium	CKV_AZURE_4	AVD-AZU-0040	SNYK-CC-TF-82
12	The web app is not configured with the recent TLS encryption version.	Medium	CKV_AZURE_15	AVD-AZU-0006	SNYK-CC-TF-145
13	The web app is not configured with the recent 'HTTP Version'.	Medium	CKV_AZURE_18	AVD-AZU-0005	SNYK-CC-TF-163
14	App Service Authentication is not configured for Azure App Service.	Medium	CKV_AZURE_13	AVD-AZU-0003	SNYK-CC-TF-160
15	The web app is not configured with 'Client Certificates (Certificates from the client)'.	Medium	CKV_AZURE_17	AVD-AZU-0001	SNYK-CC-TF-162
16	Purge protection is not activated for the Key Vault.	Medium	CKV_AZURE_110	AVD-AZU-0016	SNYK-CC-AZURE-624 SNYK-CC-TF-175
17	The retention period for the Activity Log is not configured for 365 days or more.	Medium	CKV_AZURE_37	AVD-AZU-0031	SNYK-CC-TF-156
18	The Queue service doesn't have storage logging enabled for read, write and delete operations.	Medium	CKV_AZURE_33	AVD-AZU-0009	SNYK-CC-AZURE-534
19	TLS encryption used in the Storage Account is not up to date.	Medium	CKV_AZURE_44	AVD-AZU-0011	SNYK-CC-TF-149
20	For MSSQL servers, the 'Email service and co-administrators' option is not set to 'Enabled'	Medium	CKV_AZURE_27	AVD-AZU-0023	SNYK-CC-TF-170
21	The retention period for the Network Security Group Flow Log is not configured for more than 90 days.	Medium	CKV_AZURE_12	AVD-AZU-0049	SNYK-CC-TF-152
22	The standard pricing tier is not selected.	Medium	CKV_AZURE_19	AVD-AZU-0045	SNYK-CC-TF-164
23	MySQL does not utilize the latest TLS encryption version.	Medium	CKV_AZURE_54	AVD-AZU-0026	SNYK-CC-AZURE-627
24	MySQL Database Server does not have the 'Enforce SSL connection' option set to 'ENABLED'.	Medium	CKV_AZURE_28	AVD-AZU-0020	SNYK-CC-TF-146 SNYK-CC-AZURE-628
25	The PostgreSQL Database Server does not enforce SSL connections.	Medium	CKV_AZURE_29	AVD-AZU-0020	SNYK-CC-TF-147 SNYK-CC-AZURE-630
26	Connection throttling is not activated for the PostgreSQL Database Server.	Medium	CKV_AZURE_32	AVD-AZU-0021	SNYK-CC-TF-171
27	The PostgreSQL Database Server does not have the 'log_checkpoints' enabled as server parameters.	Medium	CKV_AZURE_30	AVD-AZU-0024	SNYK-CC-TF-154
28	Storage Account access does not have 'Trusted Microsoft Services' enabled.	Medium	CKV_AZURE_36	AVD-AZU-0010	SNYK-CC-TF-172
29	Network Policies are not set up for the AKS cluster.	Low	CKV_AZURE_7	AVD-AZU-0043	SNYK-CC-TF-176
30	The API Server Authorized IP Ranges feature is not enabled for AKS.	Low	CKV_AZURE_6	AVD-AZU-0041	SNYK-CC-TF-81
31	All the activities are not included in the audit profile.	Low	CKV_AZURE_38	AVD-AZU-0033	SNYK-CC-TF-157
32	'Phone number' is not configured as a security contact.	Low	CKV_AZURE_20	AVD-AZU-0046	SNYK-CC-TF-165
33	Email notifications for high-severity alerts are not activated.	Low	CKV_AZURE_21	AVD-AZU-0044	SNYK-CC-TF-166
34	PostgreSQL is not utilizing the latest TLS encryption.	Low	CKV_AZURE_147	AVD-AZU-0026	SNYK-CC-AZURE-629

6.2.3 Summary

The general benchmarking information about various IaC tools which consists of general information about the tools such its version, type, license, and code base language it supports for scans is provided in Table 10 represented below. Additionally, it provides information about the possibility of integrating those tools with various CI/CD pipelines along with specific Azure DevOps extension support available in the market. The time to execute their IaC scans against the Terragoat is also mentioned with the possible output format of each tool. Limitations of using these tools were also noted down respectively in Table 10.

Table 10. *General metrics comparison of IaC tools*

Metrics	Checkov	Snyk IaC	Tfsec
Version	3.14.6	1.1267	1.26.0
Supported IaC types	Docker, ARM, Helm charts, Terraform (GCP, OCI, Azure, AWS), Kubernetes, CloudFormation, Serverless framework.	Terraform, AWS CloudFormation, Azure Resource Manager, Kubernetes Manifests, Helm Charts	Terraform (for AWS, Azure, GCP, CloudStack, DigitalOcean, GitHub, Kubernetes, OpenStack, Oracle).
Supported cloud providers	GitHub, GitLab, Bitbucket, and Azure DevOps	GitHub, Bitbucket Cloud, Azure DevOps, Gitlab, AWS, GCP	GitHub, Bitbucket, Azure DevOps, Gitlab, CircleCI, TravisCI, Other
CI/CD integration-specific support	BitBucket, Circle CI, GitHub Actions, Kubernetes, Jenkins, GitLab Ci, Docker, Pre-Commit, OpenAI	None	GitHub Action, Azure DevOps Pipelines Tasks
License	Apache-2.0	Not OSS	MIT license
Tool type	OSS	Commercial	OSS
Output format	Github markdown, JSON, SARIF CLI, CSV, CycloneDx, Junit XML	JSON, SARIF, HTML	lovely (default), JSON, SARIF, CSV, CheckStyle, JUnit, text, GIF
Limitation	None	100 container tests per month	Migrated to Trivy. No more support in the future.
Speed of Azure pipeline scans	24 seconds	20 seconds	7 seconds
Azure DevOps extensions support	No	No	Yes
Total number of rules available	280	195	51

The analysis of general metrics in Table 10 reveals that among the three IaC tools, Checkov stands out with the most extensive support for various IaC file types and a comprehensive range of policies it can assess. All the tools have support for various known

cloud providers, while Snyk doesn't have any specific CI/CD support. Checkov and Tfsec are the OSS tools with license Apache-2.0 and MIT respectively. In contrast, Snyk is a commercial tool with 100 container tests per month as a test limitation on the free version. Tfsec was fast on completing the execution of the IaC scans on the test, but it has stopped support recently to improve Tfsec and instead moved to Trivy. Checkov produces a great amount of security smell detection. Lastly, Snyk has less support for output format for displaying the results in the Azure pipelines compared to others.

Based on these metrics, Checkov appears to be the most effective IaC tool among the three evaluated ones, as it has detected the most issues with fewer false negatives. Tfsec stands between the other two tools in terms of these metrics but still has a substantial number of false negatives. Snyk experienced the lowest in all metrics, suggesting that it missed most of the issues although its Azure rules set is greater than Tfsec. However, without the false positives and true negatives in the datasets, we lack a comprehensive understanding of how frequently the tool overlooks false flags and incorrectly identifies false issues, which is also an important aspect of performance.

7 CONCLUSION

The primary goal of this research was to perform a thorough evaluation of the SAST and IaC tools in the context of the DevSecOps in the Azure platform, with our focus on determining the strengths and limitations of tools on various metrics. In this context, it will be easier to select the right SAST and IaC based on the various metrics as per the one's requirements.

An Azure Cloud test environment was established to analyze six different tools against their benchmarks. The analysis followed a systematic methodology, standardizing scan reports into a consistent format, categorizing outcomes of each detected vulnerability by tools into various labels such as severity, True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Afterward, performance metrics were calculated for each tool. In addition, general metrics are also considered with performance metrics to analyze from different perspectives.

We aimed at minimizing the false positives for the SAST tools which results $F_{0.5}$ -score between 0.745 and 0.286. Semgrep in terms of performance metrics stands out as a better tool with an accuracy of 0.905 for detecting true vulnerabilities and ignoring false alarms. It is the simplest SAST tool to get started for smaller teams or startup companies which also produces fewer false positives. However, it is limited to 10 contributors in a team if it is integrated with Semgrep Cloud Platform and might need more effort to analyze if it's not linked with their cloud platform. Although Snyk has the fastest execution time of the pipeline of SAST scans, it stands out way behind the rest of the tools in terms of these various metrics. In addition, Sonar Cloud is the easiest one to install on the Azure DevOps but its free scans are limited to public repositories only. Semgrep detected 9 out of 10 OWASP Top 10 vulnerabilities, while Snyk and SonarCloud detected seven of them.

On the other hand, we focused on minimizing false negatives for the IaC tools. $F_{1.5}$ -score aims to reduce the false negatives which range between 0.497 to 0.95 with the Checkov getting the highest and Snyk with the lowest $F_{1.5}$ -score value. Checkov performs well with other metrics as well because it has higher sets of 280 Azure rules to detect vulnerabilities. However, it needs a laborious effort to set up of pipeline as it is dependent on Python and the Azure pipeline needs some workaround while using Python dependencies.

Although, Snyk had 195 sets of rules it failed to beat other tools in terms of metrics. Tfsec is a moderate tool and easy to get started on Azure pipeline keeping in mind that it has migrated to Trivy and has fewer sets of Azure policies.

This research contributes to the DevSecOps paradigm and SAST and IaC tool comparison, highlighting the correlation between security activities, SDLC stages, and OWASP vulnerabilities that a security activity can detect. It encompasses ways of integrating various IaC and SAST security scans in Azure pipelines. Additionally, this study provides a relation between IaC smells and OWASP Top 10 vulnerabilities and the initial framework for further exploring various other IaC tools. Lastly, the ground truths for OWASP juice shop are also presented in this study which makes it easier to evaluate different metrics for SAST tools. This research is beneficial to software development teams, security professionals, researchers, and other stakeholders interested in building and maintaining secure software applications within DevSecOps frameworks.

In this study, we performed IaC tools analysis on a benchmark named Terragoat. The results of tools analysis with this benchmark contain limitations. Firstly, it was difficult to determine False Positive (FP) and True Negative (TN) cases because the benchmark doesn't contain false alarms, and detected vulnerabilities were True Positives in all cases. The results might be different if we were able to determine False Positives which is possible with the real-world infrastructure. Secondly, Terragoat and Checkov are developed by the same vendors which makes the results inclined more towards Checkov. While, in the SAST analysis, we consider the security hotspot as well as the vulnerabilities which result in SonarCloud having more False Positive rates which impact its end analysis results. Finally, Checkov and Semgrep pipeline execution time was affected by the extra setup required for the Python with the Azure self-hosted agent.

In summary, the analysis confirms that every SAST and IaC tool has its strengths and limitations. One can select the tools based on what they are expecting. These security scans can be integrated into the DevSecOps pipelines effortlessly and can be the starting point for those interested in transforming their software development paradigm from DevOps to DevSecOps. Extending these tools' comparison with the real-world product to yield more efficient analysis could be the future work.

8 REFERENCES

- Lwakatare, L., Kuvaja, P., & Oivo, M. (2015). Dimensions of DevOps. In C. Lassenius, T. Dingsøyr, & M. Paasivaara, *Agile Processes in Software Engineering and Extreme Programming*. (pp. 212-217). Springer, Cham. doi:10.1007/978-3-319-18612-2_19
- Acunetix. (2024, April 3). Retrieved from <https://www.acunetix.com/>
- Aljabri, M., Aldossary, M., Al-Homeed, N., Alhetelah, B., Malek, A., Alotaibi, O., & Alsaqer, S. (2022). Testing and Exploiting Tools to Improve OWASP Top Ten Security Vulnerabilities Detection. *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, (pp. 797-803). doi:10.1109/CICN56167.2022.10008360
- Aljohani, M. A., & Alqahtani, S. S. (2023). A Unified Framework for Automating Software Security Analysis in DevSecOps. *2023 International Conference on Smart Computing and Application (ICSCA)*, (pp. 1-6). doi:10.1109/ICSCA57840.2023.10087568
- Alnafessah, A., Gias, A., Wang, R., Zhu, L., Casale, G., & Filieri, A. (2021). Quality-Aware DevOps Research: Where Do We Stand? *IEEE Access*, 9, 44476-44489. doi:10.1109/ACCESS.2021.3064867
- An, J. H., Wang, Z., & Inwheel, J. (2023, 05 24). A CNN-based automatic vulnerability detection. *J Wireless Com Network*, 2023(1), 41. doi:10.1186/s13638-023-02255-2
- Anastasov, M. (2022, July 15). *CI/CD pipeline: A gentle introduction*. Retrieved October 20, 2023, from Semaphore: <https://semaphoreci.com/blog/cicd-pipeline>
- Anchore. (2024, April 3). Retrieved from <https://anchore.com/>
- Angular. (2024). *Angular*. Retrieved January 22, 2024, from <https://angular.io/>
- Anjaria, D., & Kulkarni, M. (2022, 11). Effective DevSecOps Implementation: A Systematic Literature Review. *Clarivate Analytics Web of Science* (pp. 410-417). CARDIOMETRY. doi:10.18137/cardiometry.2022.24.410417
- Ansible. (2024). Retrieved January 22, 2024, from <https://www.ansible.com/>
- AppScan. (2024, April 03). Retrieved from <https://www.hcl-software.com/appscan/products/appscan-on-cloud>
- Arachni. (2024, April 3). Retrieved from <https://github.com/Arachni/arachni>
- Atlassian. (2024). Retrieved January 23, 2024, from https://trello.com/?&aceid=&adposition=&adgroup=155219799521&campaign=20401167049&creative=666778661481&device=c&keyword=trello&matchtype=e&network=g&placement=&ds_kids=p77326405224&ds_e=GOOGLE&ds_eid=700000001557344&ds_e1=GOOGLE&gad_source=1&acs_info=Zml

- Atlassian. (2024). *Jira | Issue & Project Tracking Software | Atlassian*. Retrieved January 22, 2024, from https://www.atlassian.com/software/jira?&aceid=&adposition=&adgroup=151255099843&campaign=20389338699&creative=666773251127&device=c&keyword=jira&matchtype=e&network=g&placement=&ds_kids=p77324615863&ds_e=GOOGLE&ds_eid=700000001558501&ds_e1=GOOGLE&gad_sou
- Aydos, M., Aldan, Ç., Coşkun, E., & Soydan, A. (2022, 10). Security testing of web applications: A systematic mapping of the literature. *34(9)*, 6775-6792. doi:10.1016/j.jksuci.2021.09.018
- Azure. (2023). *FTALive-Sessions/content/devops/devsecops/1-plan-develop.md at main · Azure/FTALive-Sessions*. Retrieved October 06, 2023, from GitHub: <https://github.com/Azure/FTALive-Sessions/blob/main/content/devops/devsecops/1-plan-develop.md>
- Azure Pipelines, M. L. (2024). *Deploy an Azure Pipelines agent on Linux - Azure Pipelines | Microsoft Learn*. Retrieved January 23, 2024, from <https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/linux-agent?view=azure-devops>
- Bach-Nutman, M. (2020, 12 17). Understanding The Top 10 OWASP Vulnerabilities. doi:10.48550/arXiv.2012.09960
- Beetz, F., & Simon, H. (2022, 7). GitOps: The Evolution of DevOps? *39(4)*, 70-75. doi:10.1109/MS.2021.3119106
- Bizjak, A. (2023). *Managing DevSecOps pipeline on Azure DevOps portal*. Retrieved November 10, 2023, from <https://repozitorij.unilj.si/IzpisGradiva.php?id=145136>
- Bridgecrewio. (2023). *TerraGoat - Vulnerable Terraform Infrastructure*. Retrieved July 10, 2023, from <https://github.com/bridgecrewio/terragoat>
- Build, M. (2024). Retrieved January 23, 2024, from <https://build.microsoft.com/en-US/home>
- Bundler-audit. (2024, April 3). Retrieved from <https://rubydoc.info/gems/bundler-audit/frames>
- Checkmarx. (2024, April 3). Retrieved from https://checkmarx.com/resources/application-security-testing-platform/?utm_keyword=&utm_campaign=Europe-AD-20240101-ALL_LEVEL-GOOGLE-DES-PER-SEARCH-DSA-LP-Europe&utm_source=google&utm_medium=cpc&utm_term=&hsa_acc=2852355864&hsa_cam=20892068058&hsa_grp=158
- Checkov. (2023). *Checkov*. Retrieved December 10, 2023, from Checkov: <https://www.checkov.io/>
- Chiari, M., De Pascalis, M., & Matteo, P. (2022). Static Analysis of Infrastructure as Code: a Survey. *2022 IEEE 19th International Conference on Software*

- Architecture Companion (ICSA-C)*, (pp. 218-225). doi:10.1109/ICSA-C54293.2022.00049
- Chiari, M., De Pascalis, M., & Pradella, M. (2022). Static Analysis of Infrastructure as Code: a Survey. *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, (pp. 218-225). doi:10.1109/ICSA-C54293.2022.00049
- Clair. (2024, April 3). Retrieved from <https://github.com/quay/clair>
- Contrast Security. (2024, April 3). Retrieved from <https://www.contrastsecurity.com/contrast-community-edition>
- Dagda. (2024, April 3). Retrieved from <https://github.com/eliasgranderubio/dagda>
- Demiliani, S., Malik, A., & Zaal, S. (2020). *Azure DevOps Explained*.
- DependencyCheck. (2024, April 3). Retrieved from <https://github.com/jeremylong/DependencyCheck>
- Docker. (2024). Retrieved January 23, 2024, from <https://www.docker.com/>
- Eberhard, W. (2014). *Continuous Delivery* (1st edition ed.). dpunkt.
- Ebert, C., Bajaj, D., & Weyrich, M. (2022). Testing Software Systems. *39*(4), 8-17. doi:10.1109/MS.2022.3166755
- Eby, K. (2016). *Agile Software Development, Lifecycle, Process, and Workflow*. Retrieved October 10, 2023, from [samartsheet.com: https://www.smartsheet.com/understanding-agile-software-development-lifecycle-and-process-workflow](https://www.smartsheet.com/understanding-agile-software-development-lifecycle-and-process-workflow)
- Elder, S., Zahan, N., Shu, R., Metro, M., Kozarev, V., Menzies, T., & Williams, L. (2022, 08 06). Do I really need all this work to find vulnerabilities? *27*(6), 154. doi:10.1007/s10664-022-10179-6
- Elsayed, M., & Zulkernine, M. (2019). Offering security diagnosis as a service for cloud SaaS applications. *Journal of Information Security and Applications*, *44*, 32-48. doi:10.1016/j.jisa.2018.11.006
- Falco. (2024, April 3). Retrieved from <https://github.com/falcosecurity/falco>
- Felderer, M., & Fournoret, E. (2015, 6 1). *A systematic classification of security regression testing approaches*, *3*(17), 305-319. doi:10.1007/s10009-015-0365-2
- Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R., Pretschner, A., & Memon, A. (2016). Chapter One - Security Testing: A Survey. In *Advances in Computers* (Vol. 101, pp. 1-51). Elsevier. doi:DOI: 10.1016/bs.adcom.2015.11.003
- Freeman, E., & Forsgren, N. (2019). *DevOps* (1st edition ed.). Hoboken, New Jersey: For Dummies.

- Gokarna, M. (2020). DevOps phases across Software Development Lifecycle. doi:<https://doi.org/10.36227/techrxiv.13207796.v2>
- Gonçalves, D. H. (2022). *DevSecOps for web applications: a case study*. Retrieved July 16, 2023, from <https://recipp.ipp.pt/handle/10400.22/20787>
- GrafanaLabs. (2024). Retrieved January 20, 2024, from <https://grafana.com/>
- Harbor. (2024, April 3). Retrieved from <https://goharbor.io/docs/2.0.0/administration/vulnerability-scanning/>
- HashiCorp. (2024). *Terraform*. Retrieved January 23, 2024, from <https://www.terraform.io/>
- Hemon-Hildgen, A., Lyonnet, B., Rowe, F., & Fitzgerald, B. (2022). From Agile to DevOps: Smart Skills and Collaborations. *Information Systems Frontiers*. 22., 927-945. doi:10.1007/s10796-019-09905-1
- Higuera, J.-R., Bermejo, J., Montalvo, J. A., Villalba, J., & P關ez, J. (2020). Benchmarking Approach to Compare Web Applications Static Analysis Tools Detecting OWASP Top Ten Security Vulnerabilities. 1555-1577. Retrieved 12 10, 2023, from 10.32604/cmc.2020.010885
- Hoda, R., Salleh, N., Grundy, J., & Tee, H. M. (2017). Systematic literature reviews in agile software development: A tertiary study. 60-70. doi:<https://doi.org/10.1016/j.infsof.2017.01.007>
- Ibrahim, A., Medhat, W., & Yousef, A. (2022). DevSecOps: A Security Model for Infrastructure as Code Over the Cloud. *2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*. doi:10.1109/MIUCC55081.2022.9781709
- Imtiaz, N., Thorne, S., & Williams, L. (2021). A Comparative Study of Vulnerability Reporting by Software Composition Analysis Tools., (pp. 1-11). doi:10.1145/3475716.3475769
- Jaatun, M. G., Cruzes, D., & Luna, J. (2017). DevOps for Better Software Security in the Cloud Invited Paper. Article 69, 1–6. doi:10.1145/3098954.3103172
- Jaiswal, A., Raj, G., & Singh, D. (2014, 2 14). Security Testing of Web Applications: Issues and Challenges. 88, 26-32. doi:10.5120/15334-3667
- Jasmine. (2024). Retrieved January 22, 2024, from <https://jasmine.github.io/>
- Javed, O., & Toor, S. (2021). An Evaluation of Container Security Vulnerability Detection Tools. *Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing* (pp. 95–101). Association for Computing Machinery. doi:10.1145/3481646.3481661
- Jenkins. (2024). Retrieved January 22, 2024, from <https://www.jenkins.io/>
- JUnit5. (2024). Retrieved January 22, 2024, from <https://junit.org/junit5/>

- Katalinic, B., Zeljko, S., & Kresimir, R. (2021). Best Approach to Security in Azure Devops. In *DAAAM International Scientific Book* (Vol. 20, pp. 223-230). doi:10.2507/daaam.scibook.2021.18
- Kim, D., & Vouk, M. A. (2014). A survey of common security vulnerabilities and corresponding countermeasures for SaaS. *2014 IEEE Globecom Workshops (GC Wkshps)*, (pp. 59-63). doi:10.1109/GLOCOMW.2014.7063386
- Kim, J., Kim, T., & Im, E. G. (2014). Survey of dynamic taint analysis. *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*, (pp. 269-272). doi:10.1109/ICNIDC.2014.7000307
- Koch, A. (2014). *Agile Software Development*. Retrieved from <http://ieeexplore.ieee.org.libproxy.tuni.fi/document/9106713>
- Krief, M. (2022). *Learning DevOps*. Packt Publishing.
- Kuszczyński, K., & Walkowski, M. (2023). Comparative Analysis of Open-Source Tools for Conducting Static Code Analysis. *23(18)*, 7978. doi:10.3390/s23187978
- Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A Survey of DevOps Concepts and Challenges. *ACM Comput. Surv.*, *52*. doi:10.1145/3359981
- Li, J. (2020, 7 1). Vulnerabilities Mapping based on OWASP-SANS: A Survey for Static Application Security Testing (SAST). *Annals of Emerging Technologies in Computing*, 1-8. doi:10.33166/AETiC.2020.03.001
- Lombardi, F., & Fanton, A. (2023, 06 01). From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline. *31(2)*, 619-654. doi:10.1007/s11219-023-09619-3
- Mao, R., Zhang, H., Dai, Q., Huang, H., Rong, G., Shen, H., . . . Lu, K. (2020). Preliminary Findings about DevSecOps from Grey Literature,. *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, 450-457, doi: 10.1109/QRS51102.2020.00064. doi:10.1109/QRS51102.2020.00064
- Marandi, M., Bertia, A., & Silas, S. (2023). Implementing and Automating Security Scanning to a DevSecOps CI/CD Pipeline. *2023 World Conference on Communication & Computing (WCONF)* (pp. 1-6). IEEE. doi:10.1109/WCONF58270.2023.10235015
- Marandi, M., Bertia, A., & Silas, S. (2023). Implementing and Automating Security Scanning to a DevSecOps CI/CD Pipeline. *2023 World Conference on Communication & Computing (WCONF)*. doi:10.1109/WCONF58270.2023.10235015
- Mateo Tudela, F., Bermejo Higuera, J.-R., Bermejo Higuera, J., Sicilia Montalvo, J.-A., & Argyros, M. I. (2020). On Combining Static, Dynamic and Interactive

- Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications. *10(24)*, 9919.
doi:10.3390/app10249119
- Mburano, B., & Si, W. (2018). Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark. *2018 26th International Conference on Systems Engineering (ICSEng)*, (pp. 1-6). doi:10.1109/ICSENG.2018.8638176
- MDN Web Docs. (2024). *JavaScript*. Retrieved from <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- Mend.io. (2024, April 3). Retrieved from <https://www.mend.io/>
- Merkow, M. S. (2022). *Practical security for agile and DevOps*. Boca Raton : CRC Press, Taylor and Francis.
- Meyer, M. (2014, 05). Continuous Integration and Its Tools. 12-16.
doi:10.1109/MS.2014.58
- Microsoft. (2023). *Azure DevOps*. Retrieved August 14, 2023, from <https://azure.microsoft.com/en-us/products/devops>
- Microsoft. (2024). *C#*. Retrieved January 22, 2024, from <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>
- Microsoft Azure for Students. (2024). Retrieved January 22, 2024, from <https://azure.microsoft.com/en-us/free/students>
- MITRE Corporation. (2024). Retrieved January 23, 2024, from <https://www.mitre.org/>
- Mundra, S. (2018). *Enterprise agility : being agile in a changing world* (1st edition ed.). Birmingham, UK: Packt Publishing Ltd.
- Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps: A Multivocal Literature Review. In A. M. Mas, *Software Process Improvement and Capability Determination. SPICE 2017. Communications in Computer and Information Science* (Vol. 770, pp. 17-29). Springer, Cham. doi: https://doi-org.libproxy.tuni.fi/10.1007/978-3-319-67383-7_2
- Nodejs. (2024). *Nodejs*. Retrieved January 23, 2024, from <https://nodejs.org/en>
- Open Web Analytics. (2024). Retrieved January 22, 2024, from <https://www.openwebanalytics.com/about/>
- OWASP. (2021). Retrieved September 23, 2023, from OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation: <https://owasp.org/>
- OWASP DevSecOps Guideline - V-0.2. (2023). Retrieved November 30, 2023, from OWASP Foundation: <https://owasp.org/www-project-devsecops-guideline/latest/02f-Container-Vulnerability-Scanning>
- OWASP Juice Shop. (2023). *OWASP Juice Shop*. Retrieved December 23, 2023, from <https://github.com/juice-shop/juice-shop>

- OWASP ZAP. (2024, April 3). Retrieved from <https://www.zaproxy.org/>
- Parita, J., Sharma, A., & Ahuja, L. (2018). The Impact of Agile Software Development Process on the Quality of Software Product. *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 812-815. doi:10.1109/ICRITO.2018.8748529
- Patni, S. S., & Vaidya, M. V. (2019). Survey on Web Application Vulnerability. *HELIX*, 9(3), 4941-4946. doi:10.29042/2019-4941-4946
- PortSwigger. (2024, April 3). Retrieved from <https://portswigger.net/burp>
- Progress"Chef". (2024). Retrieved January 23, 2024, from <https://www.chef.io/>
- Prometheus. (2024). Retrieved January 22, 2024, from <https://prometheus.io/>
- Puppet. (2024). Retrieved January 22, 2024, from <https://www.puppet.com/>
- Putra, A. M., & Kabetta, H. (2022). Implementation of DevSecOps by Integrating Static and Dynamic Security Testing in CI/CD Pipelines. *2022 IEEE International Conference of Computer Science and Information Technology (ICOSNIKOM)*, (pp. 1-6). doi:10.1109/ICOSNIKOM56551.2022.10034883
- Python. (2024). Retrieved January 22, 2024, from <https://www.python.org/>
- Rahman, A., & Williams, L. (2021). Different Kind of Smells: Security Smells in Infrastructure as Code Scripts. *19*(3), 33-41. doi:10.1109/MSEC.2021.3065190
- Rajapakse, R. N., Zahedi, M., & Ali Baba, M. (2021). *An Empirical Analysis of Practitioners' Perspectives on Security Tool Integration into DevOps*. Retrieved from <https://doi-org.libproxy.tuni.fi/10.1145/3475716.3475776>
- Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2022, 01). Challenges and solutions when adopting DevSecOps: A systematic review. *141*, 106700. doi:10.1016/j.infsof.2021.106700
- Rangnau, T., Buijtenen, R. V., Fransen, F., & Turkmen, F. (2022). Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines. 145-154, doi: 10.1109/EDOC49727.2020.00026.
- Reddy Konala, P. R., Kumar, V., & Bainbridge, D. (2023). SoK: Static Configuration Analysis in Infrastructure as Code Scripts. *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, (pp. 281-288). doi:10.1109/CSR57506.2023.10224925
- SAINT. (2024, April 3). Retrieved from https://my.saintcorporation.com/resources/documentation/help/saint10_help/saint_help_ten.one.html#t=introduction.html
- Sánchez-Gordón, M., & Colomo-Palacios, R. (2020). Security as Culture: A Systematic Literature Review of DevSecOps. *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (pp. 266–269). New York, NY, USA: Association for Computing Machinery. doi:10.1145/3387940.3392233

- Sane, P. (2021). Is the OWASP Top 10 List Comprehensive Enough for Writing Secure Code? *Proceedings of the 2020 International Conference on Big Data in Management* (pp. 58–61). New York, NY, USA: Association for Computing Machinery. doi:10.1145/3437075.3437089
- Schieferdecker, I., Grossmann, J., & Schneider, M. (2012, 2 27). Model-Based Security Testing. 1-12. doi:10.4204/EPTCS.80.1
- Selenium. (2024). Retrieved January 22, 2024, from <https://www.selenium.dev/>
- Semgrep. (2024). Retrieved January 18, 2024, from <https://semgrep.dev/docs/>
- Senapathi, M., Buchan, J., & Osman, H. (2018). DevOps Capabilities, Practices, and Challenges: Insights from a Case. doi:10.1145/3210459.3210465
- Setiawan, H., Erlangga, L. E., & Baskoro, I. (2020). Vulnerability Analysis Using The Interactive Application Security Testing (IAST) Approach For Government X Website Applications. *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*, (pp. 471-475). doi:10.1109/ICOIACT50329.2020.9332116
- Slack. (2024). Retrieved January 23, 2024, from <https://slack.com/>
- Snyk. (2023). *Snyk - Developer security: Develop fast. stay.* Retrieved December 23, 2023, from Snyk: <https://snyk.io/>
- SonarCloud. (2023). *SonarCloud*. Retrieved December 23, 2023, from <https://docs.sonarsource.com/sonarcloud/>
- SonarQube. (2024, April 4). Retrieved from <https://www.sonarsource.com/open-source-editions/sonarqube-community-edition/>
- Source code analysis tools - OWASP. (2023). Source code analysis tools. Retrieved December 6, 2023, from https://owasp.org/www-community/Source_Code_Analysis_Tools
- Srokosz, M., Rusinek, D., & Ksiezopolski, B. (2018). A new WAF-based architecture for protecting web applications against CSRF attacks in malicious environment., (pp. 391-395).
- Synopsys. (2024, April 3). Retrieved from <https://www.synopsys.com/software-integrity/security-testing/interactive-application-security-testing.html>
- Taley, D. S., & Pathak, B. (2020, 09 04). Comprehensive Study of Software Testing Techniques and Strategies: A Review. *V9(08)*, IJERTV9IS080373. doi:10.17577/IJERTV9IS080373
- Tenable. (2024, April 3). Retrieved from <https://www.tenable.com/products/nessus>
- tfsec. (2023). *Aqua tfsec*. Retrieved December 6, 2023, from <https://aquasecurity.github.io/tfsec/v1.28.1/>
- Travis-CI. (2024). *Home -Travis-CI*. Retrieved January 23, 2024, from <https://www.travis-ci.com/>

- TypeScript. (2024). Retrieved January 15, 2024, from <https://www.typescriptlang.org/>
- Uddin, A., & Anand, A. (2019, 1 1). Importance of Software Testing in the Process of Software Development. 2321-0613. Retrieved December 1, 2023, from https://www.researchgate.net/publication/331223692_Importance_of_Software_Testing_in_the_Process_of_Software_Development
- Viitasuo, E. (2020). *Adding security testing in devops software development with continuous integration and continuous delivery practices*. Retrieved September 23, 2023, from <https://urn.fi/URN:NBN:fi:amk-2020060316773>
- Yarlagadda, & Teja, R. (2021). DevOps and Its Practices (March 3, 2021). *International Journal of Creative Research Thoughts (IJCRT)*, 9(3 March 2021). Retrieved September 23, 2023, from <http://www.ijcrt.org/papers/IJCRT2103016.pdf>

9 APPENDICES

9.1 Appendix A: Ground Truths Present in OWASP Juice Shop

SN	OWASP Juice Shop Category	OWASP Vulnerability Category	Filename	Line number(s)
1	Sensitive Data Exposure	A01:2021-Broken Access Control	server.ts	242, 243, 251, 253, 309
2	Sensitive Data Exposure	A01:2021-Broken Access Control	server.ts	664
3	Broken Access Control	A01:2021-Broken Access Control	frontend/src/app/app.routing.ts	48
4	Unvalidated Redirects	A01:2021-Broken Access Control	lib/insecurity.ts	132, 133, 134, 144
5	Broken Access Control	A01:2021-Broken Access Control	routes/updateProductReviews.ts	16, 18, 20
6	Security through Obscurity	A04:2021-Insecure Design	frontend/src/app/app.routing.ts	210, 245
7	Broken Authentication	A07:2021-Identification and Authentication Failures	models/user.ts	77
8	XML External Entities (XXE)	A05:2021-Security Misconfiguration	routes/fileupload.ts	80
9	Improper Input Validation	A03:2021-Injection	server.ts	466
10	Cross Site Scripting (XSS)	A03:2021-Injection	frontend/src/app/search-result/search-result.component.html	13
11	Cross Site Scripting (XSS)	A03:2021-Injection	frontend/src/app/search-result/search-result.component.ts	126, 152
12	Injection	A03:2021-Injection	routes/login.ts	36
13	Injection	A03:2021-Injection	routes/search.ts	23
14	Cross Site Scripting (XSS)	A03:2021-Injection	frontend/src/app/product-details/product-details.component.html	16
15	Cross Site Scripting (XSS)	A03:2021 - Injection	frontend/src/app/administration/administration.component.ts	51, 55
16	Cross Site Scripting (XSS)	A03:2021- Injection	frontend/src/app/administration/administration.component.html	26, 58, 90
17	Cross Site Scripting (XSS)	A03:2021-Injection	frontend/src/app/feedback-details/feedback-details.component.html	19
18	Cross Site Scripting (XSS)	A03:2021-Injection	frontend/src/app/about/about.component.ts	85

Appendix A: OWASP Juice Shop ground truth, extended from (OWASP Juice Shop, 2023)

9.2 Appendix B: Ground Truths present in Terragoat

SN	Issue	Checkov Check ID	No. of Occurrence(s)	Severity
1	SSH access does not restrict public access from the internet.	CKV_AZURE_10	1	Critical
2	RDP access is possible by anyone from the internet	CKV_AZURE_9	1	Critical
3	Custom subscription owner roles have been defined.	CKV_AZURE_39	1	Critical
4	The expiration date is not specified for all keys.	CKV_AZURE_40	1	Critical
5	Not all secrets have an expiration date.	CKV_AZURE_41	1	Critical
6	The auditing setting is not activated for SQL servers.	CKV_AZURE_23	8	High
7	The auditing retention policy for the SQL servers is not more than 90 days.	CKV_AZURE_24	8	High
8	The encryption of critical data with a Customer Managed Key in the storage is not enabled.	CKV2_AZURE_1	2	High
9	RBAC remains disabled on AKS clusters.	CKV_AZURE_5	1	High
10	Basic authentication is utilized by Azure Instance (Instead consider using SSH Key).	CKV_AZURE_1 CKV_AZURE_178	1	High
11	'All' is not selected for the 'Threat Detection types' setting.	CKV_AZURE_25	8	High
12	Alert notifications are not set up for MSSQL servers.	CKV_AZURE_26	2	High
13	Azure-managed disk's encryption is disabled.	CKV_AZURE_2	1	High
14	The 'Secure transfer required' feature is disabled.	CKV_AZURE_3	1	High
15	Azure Key Vault does not disable public network access.	CKV_AZURE_189	1	High
16	Storage blob access is accessible by anyone from the internet.	CKV_AZURE_190	2	High

Appendix B.1: Ground Truths in Terragoat with Critical and High Severity, extended from (Bridgecrewio, 2023)

SN	Issue	Checkov Check ID	No. of Occurrence(s)	Severity
1	Azure Monitoring configuration is not set up for AKS logging.	CKV_AZURE_4	1	Medium
2	The web app does not utilize the latest TLS encryption version.	CKV_AZURE_15	1	Medium
3	FTP deployments are not disabled.	CKV_AZURE_78	2	Medium
4	The web app is not configured with the recent 'HTTP Version'.	CKV_AZURE_18	2	Medium
5	App Service Authentication is not configured for Azure App Service.	CKV_AZURE_13	2	Medium
6	The web app is not configured with 'Client Certificates (Certificates from the client)'.	CKV_AZURE_17	2	Medium
7	The App service is not registered with Azure Active Directory.	CKV_AZURE_16	2	Medium
8	HTTP traffic is not automatically redirected to HTTPS for the web app in Azure App Service.	CKV_AZURE_14	1	Medium
9	The virtual machines have additional extensions installed.	CKV_AZURE_50	2	Medium
10	Key vault does not allow configuration of firewall rules.	CKV_AZURE_109	1	Medium
11	The key vault does not have recovery capabilities.	CKV_AZURE_42	1	Medium
12	Purge protection is not enabled for the key vault.	CKV_AZURE_110	1	Medium
13	The retention period for the Activity Log is not configured for 365 days or more.	CKV_AZURE_37	1	Medium
14	Access denial is not the default setting for network access rules in Storage Accounts.	CKV_AZURE_35	2	Medium
15	The Queue service doesn't have storage logging enabled for read, write and delete operations.	CKV_AZURE_33	2	Medium
16	TLS encryption used in the Storage Account is not up to date.	CKV_AZURE_44	2	Medium
17	MSSQL does not utilize the latest TLS encryption.	CKV_AZURE_52	7	Medium
18	For MSSQL servers, the 'Email service and co-administrators' option is not set to 'Enabled'.	CKV_AZURE_27	8	Medium
19	The retention period for the Network Security Group Flow Log is not configured for more than 90 days.	CKV_AZURE_12	1	Medium
20	The standard pricing tier is not selected.	CKV_AZURE_19	1	Medium
21	Email notifications for high-severity alerts are not enabled.	CKV_AZURE_22	1	Medium
22	MySQL does not utilize the latest TLS encryption version.	CKV_AZURE_54	1	Medium
23	The MySQL Database Server does not enforce SSL connections.	CKV_AZURE_28	1	Medium
24	The PostgreSQL Database Server doesn't enforce SSL connections.	CKV_AZURE_29	1	Medium
25	Connection throttling is not activated for the PostgreSQL Database Server.	CKV_AZURE_32	1	Medium
26	The PostgreSQL Database Server does not have the 'log_checkpoints' enabled as server parameters.	CKV_AZURE_30	1	Medium
27	Storage Account access does not have 'Trusted Microsoft Services' enabled.	CKV_AZURE_36	1	Medium

Appendix B.2: Ground Truths in Terragoat with Medium Severity, extended from (Bridgecrewio, 2023)

SN	Issue	Checkov Check ID	No. of Occurrence(s)	Severity
1	AKS does not integrate with Azure Policies Add-on.	CKV_AZURE_116	1	Low
2	Disabling the Kubernetes Dashboard has not been configured.	CKV_AZURE_8	1	Low
3	The disk encryption set has not been utilized by AKS.	CKV_AZURE_117	1	Low
4	AKS doesn't enable private clusters.	CKV_AZURE_115	1	Low
5	Disabling the AKS local admin account has not been configured.	CKV_AZURE_141	1	Low
6	The Network Policy is not set up for the AKS cluster.	CKV_AZURE_7	1	Low
7	The API Server Authorized IP Ranges feature is not enabled for AKS.	CKV_AZURE_6	1	Low
8	App Services do not have a managed identity provider enabled.	CKV_AZURE_71	2	Low
9	The app services do not utilize Azure Files	CKV_AZURE_88	2	Low
10	The .NET Framework version used by the web app is outdated.	CKV_AZURE_80	2	Low
11	The app service does not set the detailed error messages to on.	CKV_AZURE_65	2	Low
12	The HTTP logging for the App service remains disabled.	CKV_AZURE_63	2	Low
13	The failed request tracing for the App service is disabled.	CKV_AZURE_66	2	Low
14	The password authentication for the Virtual machine is enabled.	CKV_AZURE_149	1	Low
15	Windows VM disables encryption.	CKV_AZURE_151	1	Low
16	The key vault key is not secured by the HSM.	CKV_AZURE_112	1	Low
17	The "content_type" attribute is not defined for the Key Vault secrets.	CKV_AZURE_114	1	Low
18	Not all the activities are captured by the audit profile.	CKV_AZURE_38	1	Low
19	The SQL server allows public network access.	CKV_AZURE_113	7	Low
20	'Phone number' is not configured as a security contact.	CKV_AZURE_20	1	Low
21	Email notifications for high-severity alerts are not activated.	CKV_AZURE_21	1	Low
22	The Threat detection feature is not turned on for the MySQL server.	CKV_AZURE_127	1	Low
23	Geo-redundant backups are not turned on for the MySQL server.	CKV_AZURE_94	1	Low
24	MySQL servers have public network access enabled.	CKV_AZURE_53	1	Low
25	PostgreSQL is not utilizing the latest TLS encryption.	CKV_AZURE_147	1	Low
26	Enabling infrastructure encryption is not configured for the PostgreSQL server.	CKV_AZURE_130	1	Low
27	Enabling Threat detection policy is not configured for the PostgreSQL server.	CKV_AZURE_128	1	Low
28	Enabling geo-redundant backups for the PostgreSQL server is not configured.	CKV_AZURE_102	1	Low
29	PostgreSQL server allows public network access.	CKV_AZURE_68	1	Low
30	Customer-managed key encryption for managed disks does not utilize a specific set of disk encryption keys.	CKV_AZURE_93	1	Low
31	Enabling WAF is not configured by Application Gateway.	CKV_AZURE_120	1	Low
32	Azure Active Directory Admin is not set up.	CKV2_AZURE_7	1	Low
33	Enabling customer-managed keys for encryption is not configured for the MySQL server.	CKV2_AZURE_16	1	Low
34	Enabling customer-managed keys for encryption is not configured for Storage accounts.	CKV2_AZURE_18	2	Low
35	Base64 High Entropy String	CKV_SECRET_6	1	Low

Appendix B.3: Ground Truths in Terragoat with Low Severity, extended from (Bridgecrewio, 2023)

9.3 Appendix C: Azure YAML Configuration files

```

1 trigger: none
2
3 pool:
4   name: 'default'
5
6 stages:
7   - stage: sast_test
8     displayName: SAST Scan
9     jobs:
10      - job: SonarCloudScan
11        displayName: 'SonarCloud Code Security Scan'
12
13        steps:
14          - checkout: self
15            fetchDepth: 0
16
17          - task: SonarCloudPrepare@1
18            inputs:
19              SonarCloud: 'SonarCloud'
20              organization: 'toolanalysis-msc-thesis'
21              scannerMode: 'CLI'
22              configMode: 'manual'
23              cliProjectKey: 'ToolAnalysis_ToolAny'
24              cliProjectName: 'ToolAny'
25              cliSources: '.'
26
27          - task: SonarCloudAnalyze@1
28            inputs:
29              jdkversion: 'JAVA_HOME'
30
31          - task: SonarCloudPublish@1
32            inputs:
33              pollingTimeoutSec: '300'
34
35

```

Appendix C.1: SonarCloud Azure Pipeline yml configuration file

```

1 trigger: none
2
3 pool:
4   name: 'default'
5
6 parameters:
7   - name: pythonVersion
8     displayName: 'Python version'
9     default: '3.10.12'
10
11 stages:
12   - stage: sast_test
13     displayName: SAST Scan
14     jobs:
15      - job: SemgrepScan
16        displayName: 'Semgrep Code Security Scan'
17
18        steps:
19          - checkout: self
20            clean: true
21            fetchDepth: 0
22            persistCredentials: true
23
24          - script: |
25            mkdir -p $(Agent.ToolsDirectory)/Python/$( parameters.pythonVersion )/x64
26            mkdir -p $(Agent.ToolsDirectory)/Python/$( parameters.pythonVersion )/x64.complete
27            displayName: 'Create Python local folders'
28
29          - script: |
30            curl -LO https://github.com/actions/python-versions/releases/download/3.10.12-5200619051/python-3.10.12-linux-22.04-x64.tar.gz
31            tar -xvf python-3.10.12-linux-22.04-x64.tar.gz
32            workingDirectory: $(Agent.ToolsDirectory)/Python/$( parameters.pythonVersion )/x64
33            displayName: 'Downloaded'
34
35          - task: UsePythonVersion@0
36            inputs:
37              versionSpec: $( parameters.pythonVersion )
38              disableDownloadFromRegistry: true
39              architecture: 'x64'
40            displayName: 'Install python $( parameters.pythonVersion )'
41
42          - script: |
43            python -m pip install --upgrade pip
44            pip install semgrep==1.55.1
45            displayName: 'Install SemgrepScan'
46
47          - script: |
48            semgrep ci
49            displayName: 'Run Semgrep SAST Scan'
50
51          env:
52            SEMGREP_PR_ID: $(System.PullRequest.PullRequestNumber)
53            SEMGREP_COMMIT: $(Build.SourceVersion)
54            SEMGREP_REPO_NAME: $(SEMGREP_REPO_NAME)
55            SEMGREP_REPO_URL: $(SEMGREP_REPO_URL)
56            SEMGREP_BRANCH: $(SEMGREP_BRANCH)
57            SEMGREP_APP_TOKEN: $(SEMGREP_APP_TOKEN)

```

Appendix C.2: Semgrep Azure pipeline yml configuration file

```

1 trigger: none
2
3 pool:
4   name: 'default'
5
6 stages:
7   - stage: sast_test
8     displayName: SAST Scan
9     jobs:
10      - job: SnykSecurityScan
11        displayName: 'Snyk Code Security Scan'
12
13        steps:
14          - checkout: self
15            fetchDepth: 0
16
17          - task: NodeTool@0
18            inputs:
19              versionSpec: '16.x'
20              displayName: 'Install Node.js'
21
22          - script: |
23              npm install -g snyk@1.1267.0
24              displayName: 'Install snyk'
25
26          - script: |
27              snyk config set api=$(SNYK_TOKEN)
28
29              snyk code test --json
30
31              displayName: 'Snyk test'
32

```

Appendix C.3: Snyk Azure pipeline yml file for SAST scans

```

1 trigger: none
2
3 pool:
4   name: 'default'
5
6 parameters:
7   - name: pythonVersion
8     displayName: 'Python version'
9     default: '3.10.12'
10
11 stages:
12   - stage: iac_test
13     displayName: IaC Security Scan
14     jobs:
15      - job: CheckovSecurityScan
16        displayName: 'Checkov IaC Security Scan'
17
18        steps:
19          - checkout: self
20            fetchDepth: 0
21
22          - script: |
23              mkdir -p $(Agent.ToolsDirectory)/Python/$( parameters.pythonVersion )/x64
24              mkdir -p $(Agent.ToolsDirectory)/Python/$( parameters.pythonVersion )/x64.complete
25              displayName: 'Create Python local folders'
26
27          - script: |
28              curl -LO https://github.com/actions/python-versions/releases/download/3.10.12-5200619051/python-3.10.12-linux-22.04-x64.tar.gz
29              tar -xvf python-3.10.12-linux-22.04-x64.tar.gz
30              workingDirectory: $(Agent.ToolsDirectory)/Python/$( parameters.pythonVersion )/x64
31              displayName: 'Downloaded'
32
33          - task: UsePythonVersion@0
34            inputs:
35              versionSpec: $( parameters.pythonVersion )
36              disableDownloadFromRegistry: true
37              architecture: 'x64'
38              displayName: 'Install python $( parameters.pythonVersion )'
39
40          - script: |
41              sudo apt install python
42              pip install checkov==3.1.46
43
44              checkov -d ./terraform/azure --output junitxml > $(System.DefaultWorkingDirectory)/Test-checkov-report.xml
45              displayName: 'Install Checkov'
46
47          - task: PublishTestResults@2
48            inputs:
49              testResultsFormat: 'JUnit'
50              testResultsFiles: $(System.DefaultWorkingDirectory)/Test-checkov-report.xml
51              condition: always()

```

Appendix C.4: Checkov Azure pipeline yml configuration file

```

1 trigger: none
2
3 pool:
4   name: 'default'
5
6 stages:
7   - stage: iac_test
8     displayName: IaC Security Scan
9     jobs:
10    - job: SnykSecurityScan
11      displayName: 'Snyk Security Scan'
12
13      steps:
14        - checkout: self
15          fetchDepth: 0
16
17        - task: NodeTool@0
18          inputs:
19            versionSpec: '16.x'
20            displayName: 'Install Node.js'
21
22        - script: |
23            npm install -g snyk@1.1267.0
24            displayName: 'Install snyk'
25
26        - script: |
27            snyk config set api=$(SNYK_TOKEN)
28
29            snyk iac test ./terraform/azure --json
30            displayName: 'snyk test'

```

Appendix C.5: Snyk Azure pipeline configuration yml file for IaC scans

```

1 trigger: none
2
3 pool:
4   name: 'default'
5
6 stages:
7   - stage: tfsec_iac_test
8     displayName: IaC Security Scan
9     jobs:
10    - job: tfsecScan
11      displayName: 'tfsec IaC Security Scan'
12
13      steps:
14        - checkout: self
15          fetchDepth: 0
16
17        - task: tfsec@1
18          inputs:
19            dir: ./terraform/azure

```

Appendix C.6: Tfsec Azure pipeline yml configuration file