

---

# A Framework of DevSecOps for Software Development Teams

---

Master's Degree Programme in Information and Communication Technology  
Department of Computing, Faculty of Technology  
Master of Science in Technology Thesis  
Networked Systems Security  
June 2023  
Dinesh Sapkota

Supervisors:  
Tahir Mohammad  
Petri Sainio

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin Originality Check service.

UNIVERSITY OF TURKU  
Department of Future Technologies

DINESH SAPKOTA: A Framework of DevSecOps for Software Development Teams

Master of Science in Technology Thesis, 83 pages.  
Networked Systems Security  
June 2023

---

This master's thesis explores a broad evaluation of automated security testing in the context of DevOps practices. The primary objective of this study is to propose a framework that facilitates the seamless integration of security scanning tools within DevOps practices. The thesis will focus on examining the existing set of tools and their effective integration into fully automated DevOps CI/CD pipelines.

The thesis starts by examining the theoretical concepts of DevOps and provides guidelines for integrating security within DevOps methodologies. Furthermore, it assesses the current state of security by analysing the OWASP Web API top 10 security vulnerability list and evaluating existing security automation tools. Additionally, the research investigates the performance and efficacy of these tools across various stages of the SDLC and investigates ongoing research and development activities.

A fully automated DevOps CI/CD pipeline is implemented to integrate security scanning tools, enforcing complete security checks throughout the SDLC. Azure DevOps build and release pipelines, along with Snyk, were used to create a comprehensive automated security scanning framework. The study considerably investigates the integration of these security scanning tools and assesses their influence on the overall security posture of the developed applications. The finding of the study reveals that security scanning tools can be efficiently integrated into fully automated DevOps practices. Based on the results, recommendations are provided for the selection of suitable tools and techniques to achieve a DevSecOps practice.

In conclusion, this thesis provides valuable insights into security integration in DevOps practices, highlighting the effectiveness of security automation tools. The research also recommends areas for further improvements to meet the industry's evolving requirements.

Keywords: SDLC, DevOps, Continuous Integration (CI), Continuous Development (CD), DevSecOps, SAST, DAST, OSS Vulnerability Scanner

Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background.....	2
1.2	Problem Statement.....	3
1.3	Research Question.....	4
1.4	Research Objective.....	4
1.5	Thesis Structure .....	5
<b>2</b>	<b>Theoretical Background</b>	<b>6</b>
2.1	DevOps .....	6
2.1.1	Continuous Integration	7
2.1.2	Continuous Delivery	8
2.1.3	Infrastructure as Code	10
2.1.4	Monitoring and Logging	11
2.1.5	Agile Practices	12
2.2	DevSecOps .....	12
2.2.1	DevSecOps Guidelines	13
2.2.1.1	Planning and Design	13
2.2.1.2	Development	14
2.2.1.3	Testing	14
2.2.1.4	Deployment	15
2.2.1.5	Operations and Maintenance	15
2.2.2	DevSecOps Challenges	16
2.3	OWASP Top 10 .....	16
2.3.1	Broken Object Level Authorization	17
2.3.2	Broken User Authentication	17
2.3.3	Excessive Data Exposure	18
2.3.4	Lack of Resource and Rate Limiting	18
2.3.5	Broken Function Level Authorization	19
2.3.6	Mass Assignment	19
2.3.7	Security Misconfiguration	20
2.3.8	Injection	20
2.4	Security Automation Tools.....	21
2.4.1	Static Application Security Testing	22
2.4.2	Dynamic Application Security Testing	22
2.4.3	Open-Source Software Vulnerability Scanner	23
2.4.4	Infrastructure as Code Security Testing	24
2.5	Related Work.....	25
<b>3</b>	<b>Methodology</b>	<b>27</b>
3.1	Applications Development.....	29
3.2	Production Applications Azure Infrastructures.....	30
3.3	Infrastructure as Code in Azure .....	30
3.3.1	Azure Boards	33
3.3.2	Azure Artifacts	33
3.3.3	Azure Test Plans	34
3.3.4	Azure Repos	34
3.3.5	Azure Pipelines	34
3.4	Security Integration in Azure DevOps.....	35
3.4.1	Snyk	36
3.4.1.1	Snyk Open Source	37
3.4.1.2	Snyk Code	37

3.4.2	GitGuardian	38
<b>4</b>	<b>Implementation</b>	<b>39</b>
4.1	Test Projects .....	39
4.1.1	Production Test Application	39
4.1.1.1	Web API	40
4.1.1.2	Console Application	40
4.1.2	Infrastructure as Code Implementation	42
4.1.2.1	Parameters	42
4.1.2.2	Resources	44
4.1.2.3	Outputs	46
4.1.3	Vulnerable Test Project	47
4.1.3.1	Broken Object Level Authorization Vulnerability	47
4.1.3.2	Broken User Authentication Vulnerability	48
4.1.3.3	Excessive Data Exposure Vulnerability	49
4.1.3.4	Lack of Resource and Rate Limiting Vulnerability	50
4.1.3.5	Broken Function Level Authorization Vulnerability	51
4.1.3.6	Mass Assignment Vulnerability	52
4.1.3.7	Security Misconfiguration Vulnerability	52
4.1.3.8	Injection Vulnerability	53
4.2	Azure DevOps and CI/CD Pipeline .....	54
4.2.1	Continuous Integration using Azure Build Pipelines	55
4.2.2	Continuous Deployment using Azure Release Pipelines	60
4.3	Shifting Security to the Left .....	63
4.3.1	Security Scanning in IDE	64
4.3.2	Security Scanning in SCM System	67
4.3.3	Security Scanning in Build Pipeline	70
4.3.4	Vulnerability Scanning in IaC	74
<b>5</b>	<b>Results and Evaluation</b>	<b>76</b>
<b>6</b>	<b>Conclusion</b>	<b>81</b>
6.1	Future Works .....	83
	<b>References</b>	<b>84</b>

## List of Figures

Figure 3.1: Flow chart for security integration in DevOps .....	28
Figure 3.2: Resources in the Development Environment Resource Group .....	30
Figure 3.3: Azure DevOps Services and its software development flow [57].....	32
Figure 4.1: Code Map of event log service and queue service implementation .....	40
Figure 4.2: Code Map of Queue Receiver .....	41
Figure 4.3: Initial Development Flow .....	54
Figure 4.4: Complete Flow Until Build Pipeline .....	56
Figure 4.5: Complete CD pipeline Execution Flow .....	60
Figure 4.6: Release pipeline implementation in ADO .....	62
Figure 4.7: Release pipeline execution flow in ADO .....	63
Figure 4.8: Code Scanning and Pull Request Flow.....	64
Figure 4.9: Open-Source vulnerabilities discovered in the test project. ....	65
Figure 4.10: Code Security Vulnerabilities discovered in the test project.....	66
Figure 4.11: Security scan and execution flow in GitHub .....	68
Figure 4.12: GitHub pulls request execution flow. ....	68
Figure 4.13: Pull Requests dependency security scan details. ....	69
Figure 4.14: Privilege escalation vulnerability discovered in GitHub PR checks. ....	70
Figure 4.15: Snyk security test in CI pipeline.....	71
Figure 4.16: CI pipeline execution flow in Azure.....	72
Figure 4.17: Brief information about security issues in azure build pipeline .....	73
Figure 4.18: Vulnerability Detected - Improper Verification of Cryptographic Signature in Existing Dependency. ....	73
Figure 4.19: Snyk IaC security scan results. ....	74

## Code Snippets

Code Snippet 4.1: Template parameters used for the implementation. ....	43
Code Snippet 4.2: Production environment parameters in the parameters file.....	44
Code Snippet 4.3: Code for the deployment of the service app.....	45
Code Snippet 4.4: Server Farm and App Service outputs declarations .....	47
Code Snippet 4.5: Broken Object Level Authorization Example .....	48
Code Snippet 4.6: Broken User Authentication Example.....	49
Code Snippet 4.7: Excessive Data Exposer Example .....	50
Code Snippet 4.8: Endpoint that Lacks Resource and Rate Limit Vulnerability .....	51
Code Snippet 4.9: Broken Functional Level Authorization endpoint.....	51
Code Snippet 4.10: Endpoint with Mass Assignment flaw .....	52
Code Snippet 4.11: Studied Security Misconfiguration.....	53
Code Snippet 4.12: SQL Injection Example .....	54
Code Snippet 4.13: Trigger, agent and build configuration.....	57
Code Snippet 4.14: Pipeline restores and build configurations. ....	58
Code Snippet 4.15: Publish configuration for the project implementation.....	59
Code Snippet 4.16: Publish artifacts tasks. ....	59
Code Snippet 4.17: Snyk security test task configuration.....	72

## List of Tables

Table 5.1: Vulnerable packages discovered in the test production application. ....	77
Table 5.2: Secrets detection results of different scanning tools.....	78
Table 5.3: Vulnerabilities detected in IaC implementations.....	79
Table 5.4: Snyk code vulnerability detection results of OWASP Web API 10 list.....	80

## Abbreviations and Acronyms

ADO	Azure DevOps
API	Application Programming Interface
ARM	Azure Resource Manager
Azure AD	Azure Active Directory
Azure AD B2C	Azure Active Directory Business to Consumer
CD	Continuous Delivery
CI	Continuous Integration
CIS	Center for Internet Security
PCI-DSS	Payment Card Industry Data Security Standard
CLI	Command Line Interface
CIS Controls	Critical Security Controls
CROS	Cross-Origin Resource Sharing
CSRF	Cross-Site Request Forgery
CVE	Common Vulnerabilities and Exposure
CWE	Common Weakness Enumeration
CyRC	Synopsys Cybersecurity Research Center
DAST	Dynamic Application Security Testing
DB	Database
DoS	Denial of Service
DTO	Data Transfer Object
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaC	Infrastructure as a Code
IAST	Interactive Application Security Testing
IDE	Integrated Development Environment
IoT	Internet of Things
IT	Information Technology
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
LDAP	Lightweight Directory Access Protocol
NIST	National Institute of Standards and Technology

NoSQL	Non-SQL
NVD	National Vulnerability Database
ORM	Object Relational Mapping
OS	Operating System
OSSRA	Open-Source Security and Risk Analysis
OWASP	Open Web Application Security Project
PC	Personal Computer
PR	Pull Request
PII	Personally Identifiable Information
QA	Quality Assurance
SAST	Static Application Security Tool
SCAP	Security Content Automation Protocol
SCA	Software Composition Analysis
SCM	Source Control Management
SDLC	Software Development Life Cycle
SIEM	Security Information and Event Management
SQL	Standard Query Language
TLS	Transport Layer Security
URL	Uniform Resource Locator
VCS	Version Control System
XSS	Cross-Site Scripting
YAML	Yaml Ain't Markup Language



# Chapter 1

## 1 Introduction

According to Internet World Stats [1] until March 2022, there were 5.4 billion Internet users globally, which is 67.8% of the world population [2]. Since 2007, there has been a significant increase in the number of internet users, growing at a rate of approximately 3% [1]. This growth rate is nearly three times higher than the global population growth rate. In 2021 mobile devices contributed 54.5% of the global internet traffic [3] and it is predicted that mobile devices will contribute 55% of total IP traffic by the end of 2025 [4]. According to projections by Cybersecurity Ventures, the global internet user population is estimated to exceed 7.5 billion by 2030, including children from 6 years of age [4].

This rapid growth has resulted in an overall expansion of the IT sector. At the same time, both governmental and commercial entities are constantly increasing their resources to deliver even more online services. This rapid expansion of communication networks, digital hardware, cloud infrastructures, IoT applications, online apps, smart cars, smart cities, smart healthcare devices, and other technologies has surpassed the security industry's ability to secure them.

According to the Cyber Security Ventures report, the overall losses caused by cybercrime worldwide in 2021 reached \$6 trillion. The same source has predicted that the cost of cybercrime damage would rise at a rate of 15% per year for the next five years. There has been continuous investment in research and development to strengthen the security posture of the IT industry, which has resulted in the rapid growth of the global cybersecurity industry market value from \$3.5 billion in 2004 to \$120 billion in

# Chapter 1: Introduction

---

2017 [5]. The cyber security industry required 3.5 million security experts in 2021, but due to a shortage of skilled professionals, those positions have remained vacant [4]. These figures clearly illustrate that the current security challenges cannot be solved with the individual effort of the cybersecurity industry and security professionals. DevSecOps is an approach that was created to adopt collaboration and communication among developers, security teams, and operations teams. It involves integrating best security practices throughout the entire software development lifecycle (SLDC). The goal of DevSecOps is to deliver software that is not only fully functional and efficient but also secure. This is achieved by integrating security practices early in the development process and automating security testing.

In 2020, CyCR researchers analysed the security enforcement of existing 2,600 commercial software applications among various industries. According to their findings, 90% of target systems or apps had vulnerabilities, and 36% were critical or high-risk vulnerabilities [6]. Similarly, CyCR conducted an OSSRA examination of 2,400 commercial code bases across 17 industries in 2021. It has been reported that, on average, a codebase only consists of 22% of custom code written by the developers, and the remaining 78% of the codes were open-source dependencies. By the same report, 97% of commercial codebases contained open-source libraries, and 87% of that codebase had critical security threats [7]. Given this reality, solely having a secure custom code does not guarantee overall security. Due to the substantial presence of code in software dependencies, it becomes considerably more challenging to assess and address potential vulnerabilities.

## 1.1 Background

Typically, in DevOps practices, a small portion of code is regularly incorporated into the application's code base. Subsequently, the CI/CD pipelines, which encompass continuous integration, testing, building, and deployment preparation, are executed either manually or automatically. This process ensures the seamless integration and delivery of the application to the production environment [8], [9], [10]. Subsequently, the DevOps team forwards the application for a security audit to get approval for the production release. This last stage of security audit normally causes an unacceptable

## Chapter 1: Introduction

---

delay in software releases, and it is also too expensive to fix the issues at this stage [11]. As a result, organizations are not able to exploit DevOps CI/CD benefits. Simultaneously, there is always a potential for compromising certain security considerations in order to meet the deadlines for software release. Considering this fact, the new concept was developed to integrate security in DevOps practices with the collaboration of development, operation, and security teams. This concept emphasizes the shift of security measures from the final stage of pre-release security audits to the early stages of the Software Development Life Cycle (SDLC) [11], [12]. This approach has the potential to solve the bottleneck, facilitating rapid and cost-effective software delivery while enhancing security.

### 1.2 Problem Statement

Despite the numerous benefits, DevSecOps has its challenges to integrate security in SDLC. As previously mentioned, one significant challenge is the scarcity of security professionals, who are currently occupied with conducting security audits, consuming their valuable time. By leveraging the expertise of existing security professionals, organizations have the opportunity to establish security integration guidelines tailored to their current DevOps practices. Through the implementation of these guidelines and adopting continuous collaboration between security and development teams, the knowledge gap between the two teams can be effectively bridged. Simultaneously, developers can be empowered by utilizing automated security scanning tools that allow them to identify vulnerabilities. However, there is currently a need of recommended security scanning tools that provide developers with comprehensive coverage of key security aspects throughout the SDLC and offer actionable solutions for issue resolution.

As a result of these factors, even large organizations with ample resources find it challenging to fully automate security tests within their SDLC. In 2020, a survey conducted by CyRC and Censuswide involved 1500 IT professionals in DevSecOps roles who were working in the fields of cybersecurity and software development. According to the survey report, 33% of the respondents reported that their organization is on its way to developing mature DevSecOps practices, 30% reported that DevSecOps

## Chapter 1: Introduction

---

is used in limited scope for some specific project and the organization is expanding its use, 47% of the respondent are not using DevSecOps [7].

The data suggest that DevSecOps is not widely adopted as a common practice, and organizations with sufficient resources are creating their own approaches to meet their specific requirements. However, further research and development are necessary to automate security tests in DevOps and establish standardized best practices for automation. Only then it can become a widespread practice in companies, leading to a significant reduction in the workload of security professionals. This study will concentrate on exploring the available tools and providing recommendations on which ones would be beneficial for software developers to integrate security automation into DevOps practices.

### 1.3 Research Question

The main focus of this thesis is to investigate the feasibility of integrating automated security tests into DevOps practices for small development teams without relying on the expertise of security professionals. To address this research question, it is essential to acquire foundational knowledge about the performance of available tools by addressing sub-questions such as:

1. What is the availability of supported security automation tools suitable for DevOps?
2. How effectively do these tools cover various security aspects?
3. What is the accuracy and reliability of the security scan results they provide?
4. Can security automation be implemented early in the SDLC?
5. Will it contribute to faster software delivery and cost reduction?

### 1.4 Research Objective

The objectives of the thesis are:

1. To evaluate and assess the performance and effectiveness of different security test automation tools.
2. To identify the most promising tools and recommend for integration into DevOps practices to automate security testing.

3. To automate and provide actionable feedback at each stage of the SDLC, enabling developers to address any identified security issues promptly.

### 1.5 Thesis Structure

The study is divided into six chapters to comprehensively cover the research objectives. Chapter 2 presents the theoretical concepts necessary for understanding and implementing the proposed concept. It explores the perspective of developers in DevOps practices, discusses the guidelines for integrating security in DevOps, examines the OWASP Web API top 10 security vulnerability list and existing security automation tools, and explores ongoing research and development initiatives in this field. Chapter 3 introduces the proposed security integration conceptual model and the complete set of tools and technologies selected for the research. Chapter 4 dives into the implementation of various test projects designed to assess the current security landscape and the performance of security scanning tools. It discusses the integration of these tools into fully automated DevOps CI/CD pipelines, highlighting the concepts and security integrations in this context. The chapter also explores the performance of security tools at different stages of the SDLC. Chapter 5 evaluates the findings of the study and provides recommendations for implementing fully automated DevSecOps. Finally, Chapter 6 concludes the results of the study and presents topics for future research.

# Chapter 2

## 2 Theoretical Background

This section only focuses on the relevant concepts related to DevOps and DevSecOps, limiting the discussion to those concepts relevant to the study's context. The theoretical background section is organized into five subsections, covering the essential concepts. Subsection 2.1 addresses the relevant concepts of DevOps. Subsection 2.2 introduces the high-level concept of DevSecOps and its challenges. Subsection 2.3 covers the common security vulnerabilities discovered in 2019. Subsection 2.4 examines the concepts, strengths, and weaknesses of the available security automation tools. Finally, subsection 2.5 discusses the ongoing research and development on security automation tools.

### 2.1 DevOps

In traditional software development practices, it was quite common for the software development team to work in isolation without adequate communication within the team. Due to this inadequate communication, teams were unable to use existing knowledge within the group to its full potential. This lack of communication also resulted in misaligned objectives, additional delays in development and challenges in problem-solving. Traditionally software release cycles were long, and the software deployment process was manual, which resulted in delayed feedback, slow innovation and increased risks of bugs and errors. Due to the lack of agile methodologies traditional approach was not flexible enough to adapt to changing needs. Traditionally, stakeholders did not have enough visibility of the software systems and enough

## Chapter 2: Theoretical Background

---

feedback was not collected. All of these factors caused additional delays in software delivery [13].

Patrick Debis encountered major difficulties as a consultant while working on the data centre migration project for the Belgian government in 2007 and 2008. These difficulties were primarily caused due to the lack of communication and coordination between developers and system administrators. He explored an alternative method to address this issue as he was unsatisfied with the current software development practices [14]. Debis gathered like-minded software engineers and system administrators to a conference called "Devopsdays" in 2009, where he introduced the concept of "DevOps" [14], [9]. DevOps is a collection of practices designed to merge software development (Dev) and IT operations (Ops) teams to enhance cooperation and communication between them. It emphasises process automation through tools which enable more frequent and efficient software releases. It also prioritizes continuous improvement through continuous feedback loops [13].

The term "DevOps" has evolved through time to refer to a set of practices, culture, philosophy, and mindset utilized to accelerate the complete software development lifecycle (SDLC). It was developed to tackle weaknesses such as delayed and inefficient delivery of software, inadequate teamwork, lack of agility and flexibility, and insufficient accountability and visibility. Some of its key practices include continuous integration (CI), continuous delivery (CD), infrastructure as code (IaC), monitoring and logging, as well as lean and agile practices. These practices were crafted to improve the efficiency, reliability, and quality of software delivery [13].

### 2.1.1 Continuous Integration

The old-fashioned approach of software integration was to develop, integrate and test software components separately. Those separately developed components were integrated into a larger system later at a final stage. This led to the identification of integration problems at the final stage, which required a substantial amount of manual effort to rectify the bugs. The developers were required to manually integrate and test each component of the entire system. As the size and complexity of the system grow

## Chapter 2: Theoretical Background

---

over time, this manual integration and testing becomes more time-consuming and costly [15].

The process of automatically building and testing software changes whenever developers commit changes to a central repository is known as continuous integration (CI). The main goal of the CI process is to detect problems early in the development process so that they will be easy to detect and less expensive to fix. It requires frequent code commits to a central repository, automated building and testing, and frequent error reporting. It helps organizations in enhancing software quality, decrease the time and cost required to identify and rectify bugs and reduce the likelihood of bugs in the codebase [15], [13].

This frequent commit helps us to overcome the merge conflicts issues that appear in the traditional approach. Whenever a developer merges code changes into a central repository, it triggers an automated build process that compiles the code and runs an automated test process. This automation helps us to get instant feedback whenever new changes are merged in the codebase. In the event of a build or test failure, the errors are promptly reported to the developers, enabling them to address the issues quickly. At this point, it will be cost-efficient to fix the bugs because the whole implementation will be fresh in the developer's mind. Normally, a CI pipeline is implemented to execute multiple tasks in the specified sequence, and if a task fails, the next task will not be executed. CI pipeline is triggered whenever some changes are detected in the codebase. It extracts the most recent version of the code from the codebase and attempts to compile and build the software. Then the pipeline runs pre-configured tests on the successful test results, and pipeline execution will be successful. Additional tasks like software security scanning could also be added in CI. After the successful execution of the CI process, executable files are generated, which are later used in continuous deployment [16].

### 2.1.2 Continuous Delivery

The continuous delivery (CD) concept was developed to eliminate the manual software delivery process and prevent potential errors and misconfiguration issues associated with the traditional software release approach. On the traditional approach, due to the



## Chapter 2: Theoretical Background

---

lack of communication and coordination between the development and IT operation teams, the software delivery process used to be error-prone. The release process was a time-consuming task due to the manual release process and human errors [17].

CD advocates automating the entire software delivery process from building and testing to deployment and release. The CD process involves automatically building, testing, and packaging software changes into a format that can be deployed. Configuration files are created to deploy software to development, testing and production environments. CD process uses those configuration files to deploy software to respective environments. This approach reduces the risk of human error and enables organizations to release or update their software frequently and confidently. Organizations could also adopt A/B testing and canary releases to reduce the risk of issues and create a valuable feedback loop[18].

By adhering to the CD approach, software development teams develop software through short development cycles, assuring the reliable release of software at any given time. Normally, the IT operations team sets up a CD pipeline based on the requirements, and software developers use this pipeline to ensure the continuous delivery of their applications. The CD uses the artifacts (i.e., executable files or deployable software) build into the CI process to deploy the software in multiple (typically 2 to 5) environments. Deploying and testing applications in multiple environments ensures software quality. The CD process is constantly monitored and alerted by monitoring systems to provide continuous feedback. After the CI pipeline is successfully executed and executable files are generated, the CD pipeline is triggered to execute multiple tasks that have been defined in the pipeline. Those multiple tasks are normally the deployment of the application into different environments using preconfigured environment-specific configuration files. The pipeline executes each task in a prespecified sequence, and only once the current task has been successfully executed the pipeline moves on to the next one. In the event of errors during the execution of the pipeline, the pipeline stops and does not move on to the next task. Automatic or manual approval is usually used during the implementation of CD pipelines to prevent damage and allow human intervention as a safety net [18], [16].

## Chapter 2: Theoretical Background

---

### 2.1.3 Infrastructure as Code

IT operations teams faced challenges in provisioning and maintaining IT infrastructure due to manual, time-consuming, and error-prone configuration processes. The development of virtualization, cloud, containers, server automation, and software-defined networking was supposed to simplify these tasks but detecting and resolving issues quickly remained difficult. Configuring and updating systems was also a challenge, requiring tremendous effort for routine manual provisioning and maintenance work. Cloud Services and automation tools reduced the amount of the required changes on the infrastructure side, but applications change management complexity was not addressed. Due to that reason, IT operations teams are always required to put additional effort into tracking changes. To overcome these challenges, a team of experts was assigned to identify and develop appropriate tools and establish effective processes and procedures to address the laborious task. A more efficient and reliable way of provisioning, configuring, updating, and maintaining IT infrastructure was developed with an approach called "Infrastructure as Code" (IaC). This approach involves writing code to automate IT infrastructure deployment, making the process more efficient and reliable [19].

IaC is an IT methodology that maintains and codifies the underlying software-based IT infrastructure. Instead of manually configuring separate hardware devices and operating systems, it allows operations or development teams to automatically manage, monitor, and provision resources. The technique of IaC is sometimes referred to as programmable or software-defined infrastructure since it relies on standard software development methods [20], [21].

The goal of IaC is to establish repeatable and standardized processes for the creation and configurations of new infrastructures. Resources are defined in the definition text file, and configuration changes are made by updating the resource declaration. The resource deployment process is unsupervised and includes an extensive validation process to ensure proper deployment. Contemporary tooling can also handle infrastructure in the same way it handles software and data. This enables infrastructure management using software development tools like deployment orchestration,

## Chapter 2: Theoretical Background

---

automated testing libraries, and version control systems (VCS). Techniques like test-driven development (TDD), CI, and CD can also be used. Major companies like Amazon, Netflix, Google and Facebook are already using IaC in demanding environments for large-scale, highly reliable IT infrastructures. The goal of IaC is to make IT infrastructure support and promote changes rather than being a barrier. With IaC implementation, resource creation, configuration, update, and deployment is no longer challenging task that does not concern the operation team. Instead of doing boring, repetitive jobs all day long, the operation team could spend their time on worthwhile activities that challenge them. Additionally, IaC helps to prevent failures and enables teams to recover from infrastructure disasters rapidly and effectively [19].

### 2.1.4 Monitoring and Logging

The traditional approaches of infrastructure management had some limitations such as a lack of visibility into the underlying infrastructure, limited automation, and difficulty in identifying the root causes of issues. The conventional methods of monitoring and logging approach were not generating real-time feedback causing delays in resolving problems [13].

DevOps emphasizes continuous and automated monitoring and logging by utilizing tools that provide real-time visibility into the performance of software and infrastructure. The monitoring process constantly collects and analyses data related to the software and infrastructure performance. DevOps practices also enable real-time feedback on system performance, leveraging infrastructure and system data analysis to detect and address potential issues. DevOps advocates for continuous logging of specific actions and events that occur during various user interactions with software and infrastructure. This data can be extremely useful in identifying and resolving errors and bugs during debugging and troubleshooting. Similarly, those data will also be useful for insights into the software and infrastructure usage. That information could also be useful for security audits and the detection of unusual user behaviours or attacks [22], [13].

## Chapter 2: Theoretical Background

---

### 2.1.5 Agile Practices

Organizations were facing challenges like delayed software delivery, unaligned goals between team members and stakeholders, and a lack of collaboration between the team members. To overcome those challenges Agile software development methodology was developed [13].

The agile methodology involves breaking down large projects into smaller, manageable parts, and the complete software is delivered through multiple development iterations. This approach enables better collaboration and communication between the team members and stakeholders, which leads to improved alignment of development goals, user needs, and business requirements [23]. It also enhances project management and promotes transparency of the project's process development to all stakeholders involved. Agile practices such as daily stand-up meetings and frequent communication between development and operation teams increase transparency throughout the entire development process. Improved communication and collaboration through agile practices can reduce misunderstanding and help meet requirements [24]. However, the topic of agile practices is broad and not the focus of this study, so it is only briefly mentioned.

### 2.2 DevSecOps

Conventional security practices focus on securing applications and infrastructure at the end of the development process. However, this approach resulted in expensive and time-consuming security testing and remediation, leading to delays in releasing software to production. Traditionally, vulnerabilities were often discovered only during the final stage, making security mitigation a time-consuming and expensive process. Additionally, the separation of development, operations and security teams resulted in a lack of collaboration and communication, leading to numerous overlooked vulnerabilities [25]. Similarly, the manual process of testing and detecting security vulnerabilities was a time-consuming process that led to many vulnerabilities being missed. Furthermore, traditional security practices were not equipped to deal with the constantly evolving compliance and governance complexities [26].

## Chapter 2: Theoretical Background

---

A new methodology called DevSecOps was created to tackle these issues by integrating security into the entire DevOps process. According to [26] DevSecOps is defined as "a software development methodology that emphasizes the integration of security practices into the software development process, from planning and design stages to development and maintenance". In this methodology, secured and reliable software is developed with active collaboration between development, security, and operations teams. It advocates process automation, continuous testing, and continuous monitoring so that it will be easier to identify and mitigate security risks in real-time [27]. This implies that security is integrated throughout the entire SDLC, and teams utilize automated processes and continuous testing to detect vulnerabilities as they occur. Continuous monitoring enables the team to monitor security risks in real-time and promptly respond to any issue. Overall, the goal is to build secure and reliable software by making security an ongoing process that is integrated into each stage of development [26].

### 2.2.1 DevSecOps Guidelines

DevSecOps emphasizes implementing strategies and guidelines to develop a culture of security-centric software development within an organization. It provides guidelines for security integration into every stage of the software development process, including planning and design, development, testing, deployment, and operations and maintenance [26].

#### 2.2.1.1 Planning and Design

DevSecOps mandates the early engagement of security experts from the beginning of the project planning and design phase in SDLC. At this stage, security experts focus on identifying potential security risks based on the business requirements. Those identified risks are prioritised based on their potential impact. Security experts provide training and resources on security best practices. Organizations focus on creating a security-centric culture encouraging open communication and collaboration between different teams [28]. Security controls are implemented at the design stage, which helps to prevent and mitigate potential security risks. Organizations can conduct threat modelling to identify potential security risks and design the system to mitigate those risks. Following these guidelines, organizations can build a secure system that meets

## Chapter 2: Theoretical Background

---

their business requirements while ensuring security and compliance requirements are met [26].

### 2.2.1.2 Development

In the development phase, secure software development practices such as code reviews, static analysis, and vulnerability scanning are utilized to identify and mitigate security issues early in the development process. Organizations could utilize security experts to organize training sessions on secure coding practices, secure coding standards, secure coding techniques, and a secure development lifecycle [25], [29]. The DevSecOps guideline also stresses the importance of using automated security testing tools, such as static and dynamic analysis tools, continuously to detect vulnerabilities throughout the entire development process. It highlights the need for prompt mitigation of any security issues as they are identified, which helps to reduce the additional delays in the release process. The use of OWASP Top 10 and CIS Controls is recommended to enforce security best practices throughout the development process. By following these guidelines, organizations can ensure that security is integrated into development, thereby reducing the chances of security vulnerabilities in the final product [26].

### 2.2.1.3 Testing

The DevSecOps guideline recommends carrying out various types of tests, including unit tests, integration tests, functional tests, and tests specific to security. In addition, it recommends conducting penetration testing and vulnerability scanning as part of security testing [30]. Several security risks, such as flaws in authentication, authorization issues, vulnerabilities in input data, cross-site scripting (XSS), cross-site request forgery (CSRF), and exposure of sensitive data, must undergo testing according to the guideline. Test environments should mirror the production environment to identify security risks in the real-world setting. Security tests should be conducted throughout the SDLC, including the build and development phases. Security tests should be carried out using security testing tools and frameworks like OWASP ZAP, Burp Suite, etc. These tools help to identify and mitigate security issues. Following these guidelines organizations can ensure that their software is thoroughly tested for

## Chapter 2: Theoretical Background

---

security vulnerabilities and identified issues are mitigated before the software is deployed to production [26].

### 2.2.1.4 Deployment

The guideline recommends implementing deployment security controls during the deployment process. This can be achieved through access control mechanisms, change management procedures, and configuration management. The practice of secure configuration management ensures that all necessary systems and components are configured in a secure and up-to-date manner. Deployments should be scanned/monitored to detect security risks such as unauthorized changes and misconfigurations in the system [31]. DevSecOps approach promotes the use of deployment tools that automate the deployment process, which reduces the risks of errors and misconfigurations. Similarly, after the deployment process, post-deployment tests are conducted to verify that the deployment was successful and that all the system components are working correctly. By following these guidelines, organizations can make sure that their software deployments are secure, reliable, and operate correctly [26].

### 2.2.1.5 Operations and Maintenance

Guidelines emphasise the implementation of security monitoring and incident response procedures, which would be helpful for security incident detections in real-time. It also enforces real-time incident reporting, which could be helpful in real-time security incident response. Log management analysis tools could be used to collect and analyse logs from different systems and applications. Those analysis results are helpful in detecting potential security issues [32]. Vulnerability scans are regularly conducted to identify potential vulnerabilities in the hosted applications. This can also ensure that systems and applications contain the latest security patches. Network security controls like firewalls, intrusion prevention systems (IPS), and virtual private networks (VPNs) are implemented to secure applications from external threats. A proper access control mechanism is implemented to ensure that only authorized users would have access to sensitive data and systems. Organizations could follow these guidelines to ensure that

## Chapter 2: Theoretical Background

---

their systems and applications are monitored for security issues, vulnerabilities are identified and remediated, and sensitive data is protected from unauthorized access [26].

### 2.2.2 DevSecOps Challenges

As discussed earlier, DevSecOps requires a wide range of tools and techniques to be integrated into DevOps practices. Due to those reasons, organizations may face several challenges while implementing secure DevOps practices. Lack of security expertise in the DevOps team is the main challenge [28]. All small, medium, and big organizations lack enough security experts. Due to this reason, some organizations may not have security expertise, knowledge and skills required to effectively integrate security into the DevOps process. Similarly, there is a lack of sufficient collaboration and communication between development, operations, and security teams. DevSecOps integration requires a major change in the existing process, tools, and workflows. Due to that reason, there are hesitation and resistance to big change because it could slow down the existing development flow. Currently, there are not a significant amount of available security integration tools, and those available tools also have compatibility issues with existing systems. These issues also result in technical challenges and additional delays in DevSecOps implementations. It is challenging for organizations to develop a DevSecOps methodology that ensures regularity requirements compliance. Organizations must address these challenges to effectively integrate security practices into the DevOps process and ensure their software development practices are secure and compliant [26].

### 2.3 OWASP Top 10

The renowned non-profitable foundation called Open Web Application Security Project (OWASP) has been used as the source of the top 10 vulnerabilities. OWASP was launched on 1<sup>st</sup> December 2001 with the sole purpose of improving software security. OWASP publishes the top 10 web application security risk every 3-4 years based on the most common and critical vulnerabilities found during that period. Those items in the top 10 list are considered the minimum baseline of security enforcement while developing web applications [33]. OWASP API security top 10 list published in 2019 is



## Chapter 2: Theoretical Background

---

used to test the performance of SAST tool. All relevant items in the list will be discussed in the upcoming sections.

### 2.3.1 Broken Object Level Authorization

An access control method known as Object Level Authorization ensures that a user can only access the objects to which they are authorized. It is typically used at the code level. In this approach, the API endpoint receives an object identifier like Id in a request object and then it executes some operation on that object. Before performing any operation, a security mechanism should be enforced to check Object Level Authorization. This access check should confirm that the logged-in user is authorized to perform the desired action on the requested Object. Unauthorized users can potentially access, modify, or delete sensitive information if proper object-level authorization is not ensured. The approach to authorization and access control has improved significantly, and the latest best practices can efficiently tackle the issue of Broken Object Level Authorization. It is quite common on web APIs because the server's component does not track the client state and it fully depends on the parameters such as object Id in the request to determine which objects to access. Even with the appropriate access control and authorization in place, there is still a possibility that access checks may not be enforced before sensitive data is accessed. Normally these access control misconfigurations are not considered easily detected by SAST or DAST scanning [34].

### 2.3.2 Broken User Authentication

According to the study conducted in 2019 by OWASP, it was found that the user authentication process was not properly implemented. As a result of that, attackers could be able to compromise authentication tokens and impersonate a legitimate user identity for a short period or permanently. This compromises the ability of a system to properly authenticate users resulting in a compromised API [35] API Authentication is considered a complicated process that could cause misunderstanding about the authentication boundaries. At the same time, since API endpoints are exposed, the authentication mechanism has become an easy target for attackers. Due to that reason, authentication endpoints are vulnerable to exploits. While considering web API's authentication endpoints, it is the crux for authorization while communicating with

## Chapter 2: Theoretical Background

---

other systems, and those endpoints must be secured. According to OWASP, an authentication endpoint is not secure if it allows credential stuffing, allows brute force attacks, allows weak password, sends authentication tokens in URL, tokens authenticity is not checked, accepts unsigned tokens, uses non-encrypted or weakly signed passwords, and uses weak encryption keys [36].

### 2.3.3 Excessive Data Exposure

According to the same study, Excessive Data Exposure ended up as a third item in the list where developers are kin to expose complete object properties without considering the sensitivity of the data it holds. This also indicates that the endpoints completely depend on the vulnerable clients for data filtration before displaying them to the user [35]. The attacker could easily exploit this vulnerability by sniffing the network traffic and analysing the endpoint responses. This vulnerability has been added as a third security risk due to its severity and common practice. It often results in the disclosure of sensitive data to clients due to flawed practices. Improper segregation of data based on their sensitivity in API endpoints leads to severe vulnerabilities because it is the primary data source. This vulnerability is considered difficult to be detected by automatic security scanning tools. Because it is difficult for those tools to distinguish between legitimate and sensitive data without knowing the context of the application [37].

### 2.3.4 Lack of Resource and Rate Limiting

The same study indicates that majority of API endpoints do not have any limits on the number of requests a client can make to obtain a resource. Not having this restriction makes the API endpoint wide open to Denial of Service (DoS) attacks. If the authentication service does not enforce such restrictions, it is considered an authentication flaw that is completely open for brute force attacks [35]. It is quite common in the current practice that most existing APIs lack resource and rate-limiting restrictions. Automatic security scanning tools can more easily detect this vulnerability than the previous three vulnerabilities. API endpoints could be vulnerable without setting an appropriate limit on execution timeouts, max allocable memory, numbers of processors, request payload size, number of requests per client/resource, number of records per page to return in a single response etc. [38].

## Chapter 2: Theoretical Background

---

### 2.3.5 Broken Function Level Authorization

In the current context, APIs are bound to have a complex access policy due to different hierarchies of groups and roles. At the same time, endpoints seem to have ambiguous separation of administrative and regular functions resulting in an authorization flow, which could be exploited by the attacker to gain illegal access to sensitive user's resources and administrative functions [35]. In this context, an attacker sends a legitimate request to the endpoint to which they should not have access. Mismanagement of user groups, roles, or administrative privileges, could expose sensitive endpoints to anonymous or unprivileged users. In the context of functions, authorization is done in the configurations and in some special situations, it is also done in the code level. Enforcing a proper authorization check could be a confusing task for the developer due to the complex user hierarchy with groups and roles. By exploiting these flaws, attackers could gain access to unauthorized features because administrative functions are the target of attacks. To prevent this kind of flaw, we could manually check if a normal user could access administrative endpoints or send PUT, POST, or DELETE request. Similarly, we could also check if the user from one user group could access the resources from another user group by guessing the URL [39].

### 2.3.6 Mass Assignment

Mass assignment is usually caused due the direct binding of the data received in the request body without data sanitization. Before processing the data, the endpoint should have properties of filtration logic based on the allow list. The attacker must have some understanding of the business logic, object relations and API structure to exploit this vulnerability. Mass assignment attacks are easier in APIs because they normally expose the application business logic and properties names. Since modern frameworks could automatically bind request body into code variables and internal objects due to that reason, it is quite common in existing APIs. Attackers could use this existing mechanism to update or overwrite sensitive objects. Its exploits could cause a privilege escalation, data tempering and security bypass. This vulnerability is also considered most unlikely to be detected by automatic security scanning tools [40].

### 2.3.7 Security Misconfiguration

According to OWASP API security study, security misconfiguration has been ranked as the 7<sup>th</sup> serious flaw. Security Misconfiguration is normally caused using the default or incomplete configurations. Some of those missed configurations could be improper HTTP headers configurations, usages of unnecessary HTTP methods, anonymous allowance of CROS requests, and logging sensitive information in error messages [35]. To exploit these vulnerabilities, attackers try to find unpatched security flaws, generic sensitive endpoints, and unprotected resources to gain unauthorised access. According to the study, it is quite common in the API stack, network communication layer and application layer. Large numbers of automatic tools are available to detect security misconfigurations. Security misconfiguration could also expose sensitive internal system details that could result in server hijack [41].

### 2.3.8 Injection

This vulnerability is categorized as a situation where the input data is not validated, and those inputs are directly sent to the interpreter in the form of a command or query. It is also known as SQL, NoSQL, and command injection attack [35]. An attacker could exploit this vulnerability by sending crafted input data to the endpoint. The interpreter, such as the database, may perform unauthorized queries or command execution that are hidden in the input parameter. This could lead to the disclosure of sensitive PII, granting administrative privileges to the attacker, or manipulation or destruction of sensitive data. It has been a common vulnerability for a long time but is now declining in the top 10 list due to increased security risk awareness. However, it still remains a widespread weakness. Sensitive information disclosure and data loss are considered the main impact of injection attacks. It also has the potential of DoS and unauthorized hostile takeover of the application. It is more easily detectable by SAST tools than other vulnerabilities in the list. The leading causes for this vulnerability are lack of validation, filtering and sanitation of input data and direct concatenation of the client inputs in SQL, NoSQL or LDAP queries, OS command, XML parsers and ORM [42].

## Chapter 2: Theoretical Background

---

Improper assets management and insufficient logging and monitoring are last two vulnerabilities in OWASP API security top 10 vulnerabilities. However, they are out of the scope of this study. Therefore, they are not discussed in detail in this section.

### 2.4 Security Automation Tools

The National Institute of Standards and Technology (NIST) released Security Content Automation Protocol (SCAP) in 2009 [43]. The protocol was designed to offer a standardized approach to automate security automation tasks like vulnerability scanning, configuration assessment, and compliance checking. This standardization provides a common language for different security tools to interoperate and share data, which allows collaboration between organizations to streamline security management processes and reduce the risk of security breaches. The main goal of SCAP is to provide a standard framework for automating security-related tasks, leading to increased efficiency and effectiveness of security management. SCAP was designed to integrate key security components like Common Vulnerabilities and Exposure (CVE) dictionary, Common Platform Enumeration (CPE) dictionary, Common Configuration Enumeration (CCE) dictionary, Extensive Configuration Checklist Description Format (XCCDF) etc. SCAP standard has a certification process which evaluates different products and tools against a set of requirements. This certification ensures that the security tools fulfil the requirements and they are interoperable with other SCAP-certified tools. Some of the giant security products and tools vendors like McAfee, RedHat, IBM, tenable, RAPID7, ThreatGuard, Qualys etc. are SCAP certified [44].

SCAP covers a wide range of products and tools. However, considering tools that are helpful for the software development process, those essential tools like software composition analysis tools, static application security testing tools, dynamic application scanning tools etc., are yet to be recognized as effective and recommended. These tools cover most aspects of software security but are not platform-independent. Security analysis is not fast enough, and security scan results are not actionable to mitigate the issue. Due to those reasons, they are not widely used. Some of the available software security tests are Snyk, Synopsys Code Sight, GitHub Advanced Security(codeQL), GitGuardian, OWASP ZAP, Nessus, W3AF, Burp Suite, Nikto etc.

## Chapter 2: Theoretical Background

---

### 2.4.1 Static Application Security Testing

A study conducted by [45] states Static Application Security Testing (SAST) is a technique used to identify security vulnerabilities in an application's source code. Vulnerabilities are identified by analysing the code using predefined rules to identify common vulnerabilities, such as injection attacks, cross-site scripting (XSS) attacks, buffer overflows etc. SAST tools scan the source code without running the application. During the scan, it builds a model of the code structure that is used for the detection of potential vulnerabilities. All the potential vulnerabilities detected during the scan are reported to the developer as scan results or reports which could be mitigated by the developers. It is the most cost-efficient and effective approach because security risks are detected in the early development process [45]. Security scan report normally contains information like the location of the vulnerabilities in the source code, the severity score of the vulnerability, and possible mitigation recommendations. Developers use this report to mitigate security vulnerabilities.

Rule-based SAST tools are quite common, but they are known for generating large numbers of false positives. To overcome that issue, machine learning based SAST tools are being developed and they are more effective in the long term. Because those tools could learn from new vulnerabilities and adapt themselves to new coding patterns [45]. But they are not that attractive for small and medium size companies due to their higher initial setup cost. Existing SAST tools are known for their weakness like a large number of false positives, limited vulnerability coverage, inefficient contextual understanding, and difficulty in prioritizing result [46]. Due to all these reasons, SAST tools are not commonly used by developers to mitigate security risks [47]. Some available SAST tools are CodeQL, SonarQube, Checkmarx, Veracode, Fortify, Kiuwan, Parasoft, AppScan, Coverity, RIPS etc.

### 2.4.2 Dynamic Application Security Testing

Dynamic application security testing is a technique used to access the security of the software application while they are running in the production environment. Dynamic testing uses an automated process that continuously interacts with the application in

## Chapter 2: Theoretical Background

---

real-time probing the application's inputs and outputs to detect potential vulnerabilities. Normally it involves instrumentation, test execution, analysis, and reporting steps. In the instrumentation step, additional code (i.e., instrumentation code) is inserted into the application's executable files. This instrumentation code is used to monitor and gather data for analysis. Test execution is the second step, where the testing tool generates input data (i.e., test cases) and feeds those data to the application via a user interface of API interface. In the third step testing tool monitors tests executions and records failed executions or abnormal behaviour. Recorded information is analysed to detect security vulnerabilities in the application. As a final step, after a complete analysis of the tests, testing tools generate a report with the list of detected vulnerabilities [48].

Both SAST and DAST tools are recommended to be used together to ensure the complete security assurance of the application. Currently, DAST is more effective than the SAST tool. The strengths of DAST tool are real-world testing, rapid identifications of vulnerabilities, easy to use, scalability, and detection of configuration-related vulnerabilities. But DAST tools also has some weakness like limited code coverage, difficulty in reproducing vulnerabilities, false negatives, limited testing scenarios and performance issues [49]. Some of the available DAST tools are OWASP Zed Attack Proxy, Acunetix, WebInspect, Acunetix, AppScan, Netparker, Quals, Nmap, Vega, IronWASP etc.

### 2.4.3 Open-Source Software Vulnerability Scanner

According to Dann et al. study in 2022, open-source software (OSS) vulnerability scanners are tools specifically designed to detect vulnerabilities in OSS libraries and frameworks. Generally, it collects all the information about existing project dependencies, including direct and transitive dependencies. Then it compares the dependency information across the known vulnerability databases like the National Vulnerability Database (NVD) or Common Vulnerabilities and Exposures (CVE). Whenever it finds that the existing project dependencies are in those known vulnerability databases, those dependencies are reported as vulnerable dependencies. In some of the scanners, machine learning techniques are also used for the detection of formerly unidentified vulnerabilities. The scanner first downloads software source code

## Chapter 2: Theoretical Background

---

and then analyses it using one or more analysis techniques. During the analysis, it creates an abstract representation of the software and its dependencies. Then uses that representation for its comparison with the known dependencies vulnerabilities database and reports to the user if it finds any known vulnerabilities [50].

The existing open-source vulnerability scanner has some serious weaknesses, like limited data source, false positive and false negative, scalability, context sensitivity, limited support for non-code artifacts, and a false sense of security [51]. Currently, most of the existing OSS vulnerability scanners provide mixed results; some have a high detection rate with a high false positive rate and some have a lower false positive rate with the possibility of missing some vulnerabilities. Due to that reason, organizations are recommended to evaluate the existing scanners and use a scanner which is suitable according to the requirements [50]. Some of the recently available OSS vulnerability scanners are Sonatype Nexus IQ, OWASP Dependency-Check, Anchore, Clair, Snyk etc.

### 2.4.4 Infrastructure as Code Security Testing

Infrastructure as Code (IaC) security testing tools are used to detect and mitigate security vulnerabilities in IaC templates. IaC security testing tool analyses the code that has been used to create and manage cloud infrastructure. It considers security vulnerabilities and compliance issues during the test. This could be used while the infrastructure is under development before deployment, or it could also be used in the existing infrastructure to check if there are any potential security vulnerabilities. Security testing could also use combinations of static analysis and dynamic testing to detect security vulnerabilities and compliance issues in IaC. The tool could examine common security issues like hard-coded secrets, insecure protocols, weak authentication, authorization mechanism etc. Similarly, some of the tools also check compliance issues using industry standards like CIS benchmark and PCI-DSS. Normally most of the tools generate a report as a scan result which lists all the detected vulnerabilities and provides mitigation suggestions to help the developer to fix the identified issues [31].



## Chapter 2: Theoretical Background

---

Some of the challenges of IaC security scanning tools are lack of standardization, the complexity of IaC, integration with existing tools, lack of expertise and false positives [52]. Some of the benefits of IaC security scanning tools are improved security, early detection of issues, increased efficiency, and better compliance. Due to all these benefits, more organizations are showing interest towards the tools [31].

### 2.5 Related Work

SAST tools can be used to identify security flaws in software during the early stage of development, which can result in cost-effective software delivery. However, current SAST tools have limitations such as false positives, limited coverage of vulnerabilities, inefficient contextual understanding, and difficulty in prioritizing results. Continuous research is going on to improve the effectiveness of SAST tools because it has the potential to empower developers to ensure security compliance. Work in [47] has proposed a framework to enhance the effectiveness of SAST tools using some of the steps like pre-processing the source code and extracting the feature from processed code (e.g., syntax features, data flow features, and control flow features). After that, a machine learning algorithm is used to analyse the extracted features to identify potential vulnerabilities. The author also purposes to use supervised learning methods like decision trees and support vector machines, as well as unsupervised learning methods such as K-Means clustering, to achieve better performance [47].

It is recommended to use both SAST and DAST tools together to ensure complete security assurance of the application. Currently, DAST tools are considered more effective than SAST tools. According to [49] SAST and DAST integration into CI/CD pipeline is not a common practice because of the lack of knowledge and skills, time and resources, integration challenges, and false positives and negatives. Only 28% of organizations have integrated security testing tools in their CI/CD pipelines, and among them, only 10% have integrated DAST tools. Organizations have recognized the importance of integrating security into the development process, and there is an increasing interest in integrating security testing tools, including DAST, in CI/CD pipelines. According to the research, integrating DAST tools into CI/CD pipelines can

## Chapter 2: Theoretical Background

---

be effective if the suitable tools are integrated with proper configuration addressing the false positive issues [49].

The Open-Source (OSS) vulnerability scanner has undergone significant development. Despite some limitations, organizations could benefit from using these tools. Various tools are available that can be utilized in IDEs, repositories, and CI/CD pipelines. However, it has not been common for organizations to use OSS vulnerability scanners. According to the study, only 9% of the project that has been analysed in this study were using some OSS vulnerability scanner. All the remaining projects relied on manual code inspection or external security audits. This indicates that even though scanner has potential benefits in detecting OSS vulnerability, it has not been commonly practised yet [50].

Many organizations still use manual methods or ad hoc tools for IaC security testing, and automated IaC security scanning tools are not used in these organizations. Existing tools have some deficits, like lack of standardization in the security testing process, insufficient automation, and lack of integration into the software development process. The tool has the potential to help the developers in decreasing the security vulnerabilities that might be introduced in the cloud infrastructure [31]. However, the usage of IaC security scanning tools is not a common practice in organizations.

Despite the significant progress made in the development of security automation tools, it is not yet a widespread practice for organizations to prioritize security in the early stages of the software development life cycle. Although various tools have been developed to address this issue, their adoption rates remain low. By leveraging these tools, developers can implement security measures from the start of the SDLC. This study aims to integrate security automation tools into a fully automated DevOps CI/CD pipeline that enables developers to enforce security in small teams.

# Chapter 3

## 3 Methodology

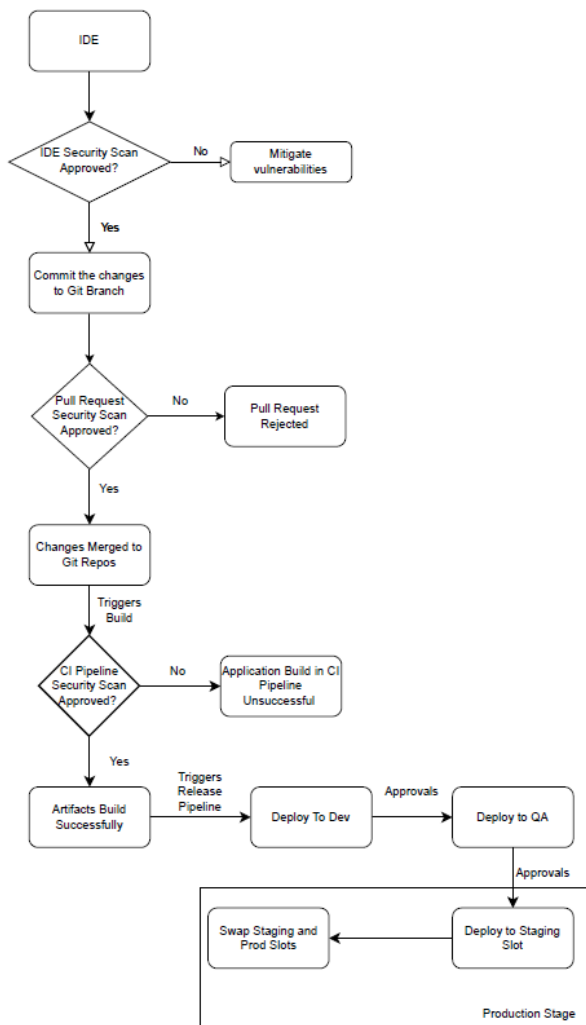
Since 2015, organizations have been adopting DevOps practices, but DevSecOps has not become common practice due to various challenges [13]. Therefore, there is a demand for security scanning tools that can be integrated into the software development life cycle, enabling developers to enforce security. The primary goal of the research was to implement a fully automated DevOps CI/CD pipeline with integrated security. The intention was to automate security tests from the initiation of code development in the IDE until deployment in the production environment. The study aims to provide developers with security scanning tools that can be integrated into the DevOps CI/CD pipeline.

To achieve this goal, the first step was to implement a fully automated DevOps CI/CD pipeline. In this process, whenever a developer merges new changes in the version control system, it triggers the CI pipeline in DevOps CI/CD automation. The CI pipeline performs all configured tasks, such as running automated tests, building and publishing artifacts. After the successful execution of the CI pipeline, the CD pipeline is activated, which publishes the application to various infrastructures according to the predefined configurations in the different release pipeline stages.

Once a fully automated DevOps CI/CD pipeline has been implemented, the next step is to integrate security scanners into each stage of the pipeline in order to empower developers to ensure security. This involves adding security scanner plugins in IDEs to scan code for vulnerabilities before pushing any changes to the repository. Additionally, Open-Source Vulnerability Scanner can be integrated into the Source Control

## Chapter 3: Methodology

Management (SCM) system to check each Pull Request (PR) before merging new changes into the master branch. Security scanners can also be integrated into the CI pipeline to detect security vulnerabilities before publishing the executable file used for application deployment. However, in this study, security integration possibilities are explored only until the CI pipeline in the DevOps flow. Detailed information on this implementation will be presented in the next chapter, and a diagram of the proposed security integration execution flow can be seen below.



**Figure 3.1: Flow chart for security integration in DevOps**

The flow diagram above shows the execution flow after integrating a security scanner into a fully automated DevOps CI/CD pipeline. In the first stage, developers can perform a security check on new features and mitigate existing vulnerabilities if the

## Chapter 3: Methodology

---

scanner detects any. After pushing the changes to the feature branch in the repository, a Pull Request (PR) is created to merge the new feature into the master branch. Before merging, the integrated security tool is triggered by the Source Control Management (SCM) system to perform a security check. If the scanner detects vulnerabilities, the PR is rejected, and if not, the PR is completed. The SCM system then triggers the CI pipeline, which extracts the source code to build the executable files and performs a security check before publishing the executable files into the drop folder. If the scanner detects vulnerabilities, the pipeline execution fails, and if not, the pipeline triggers the development CD pipeline. The development CD pipeline deploys the application into the development environment cloud infrastructure and triggers the Quality Assurance (QA) CD pipeline after the successful deployment. The QA CD pipeline requests pre-deployment approval from the developer and deploys the application into the QA environment if the approval is granted. The production CD pipeline is triggered after a successful deployment in the QA environment and sends an approval request to the manager for execution. The manager manually triggers the production CD pipeline after receiving confirmation from the QA engineer.

### 3.1 Applications Development

A computer with the Windows 10 operating system was utilized for developing applications using Visual Studio 2019 and Visual Studio Code IDEs. The objective was to investigate the rate at which newly discovered vulnerabilities emerge and how applications that are secured during development can still be vulnerable after a certain period. Therefore, an older version of the ASP.NET Core 3.1 framework in C# programming language was used for application development. The NuGet packages available in 2019 were utilized to create and test the applications. Because of its platform-independent nature, Web API applications are widely used. As a result, two Web APIs and a console application were developed for the study. A test application that uses microservice architecture was created using a combination of Web API and console application. An Infrastructure as Code (IaC) project was created using an ARM template for the deployment of the Azure infrastructure that was necessary for the test production application. A Web API with vulnerabilities was developed to include the

## Chapter 3: Methodology

top 8 OWASP API Security vulnerabilities in order to test the performance of the SAST tool.

### 3.2 Production Applications Azure Infrastructures

Azure provides a wide range of cloud services that can be used for hosting applications. For this study, the two applications were hosted on Azure cloud using Azure App Services, Azure Key Vault, Azure Active Directory, SQL Database, and Storage Account, which are commonly used in the Microsoft technology stack for production applications. As per standard best practices, it is recommended to host three different versions of an application: development, quality assurance, and production, in separate environments. In this study, the same application was deployed in Azure, with three separate resource groups for each version. All the resources for each deployment environment were created under the same resource group. This separation of resources based on the deployment environment also helps in implementing Infrastructure as Code deployment. The image below shows the various resources used in the development environment resource group.

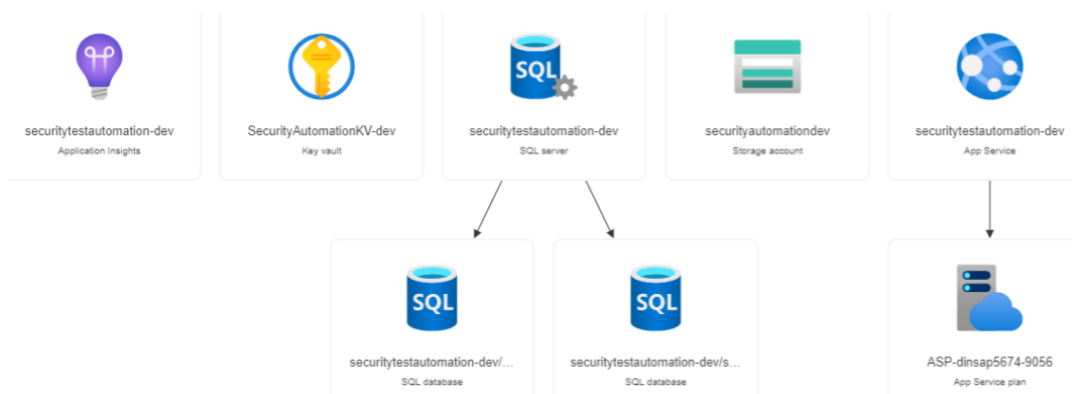


Figure 3.2: Resources in the Development Environment Resource Group

### 3.3 Infrastructure as Code in Azure

Microsoft released Azure Resource Manager in 2014 to address the challenges faced by IT operation teams due to manual, time-consuming, and error-prone configuration processes [53]. Azure Resource Manager can also be understood as an Azure deployment and management system. It offers a management layer that lets us add, modify, and remove resources in Azure subscription. Software development teams can

## Chapter 3: Methodology

---

use it to specify the required infrastructure to deliver solutions using declarative Azure resource manager (ARM) templates. Similarly, other third-party IaC tools like Terraform, Ansible, Chef, Pulumi etc., are also available [54].

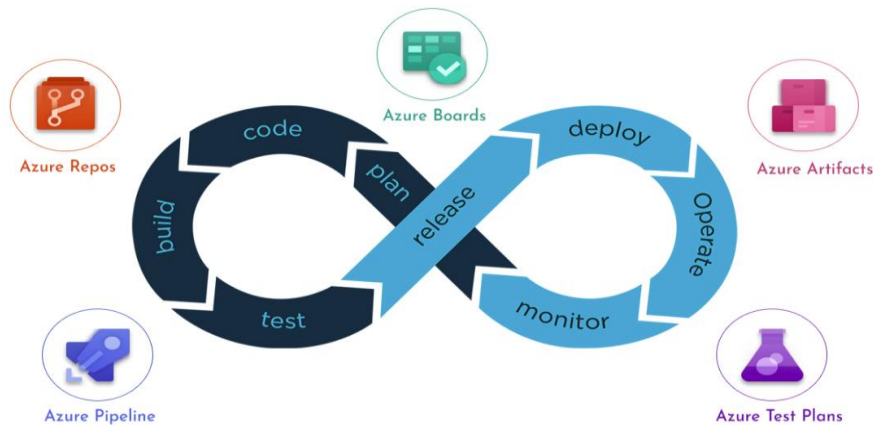
Azure Resource Manager (ARM) Templates can also be used to develop IaC in Azure. ARM template is a JSON file which normally defines the setup and settings of our project. Since this template uses declarative syntax, we can declare what we want to deploy without specifying the exact series of programming instructions that will be used to create those Azure resources. Normally the information about resources to be deployed and their associated characteristics are specified in the template [55]. The IaC method is used to deploy a range of Azure resources, such as app services, functions, SQL Server, storage accounts, networks, virtual machines, load balancers, and connection topologies. This IaC model ensures that the same environment is produced every time it is deployed, much like how the same source code generates the same binary each time [54].

After deployment, we can secure and arrange our resources using management tools like locks, tags, and access control. Resource Manager receives requests sent through Azure APIs, tools, or SDKs which are then authenticated and approved before sending those requests to the relevant Azure service [53]. Since the aim of this study was to expose the threats that production applications are facing over time ARM templates have been used for this study. Even though the more popular Bicep language was already released in 2020, Azure DevOps

Azure DevOps (ADO) offers a set of services which are essential for rapid software development and delivery. Those set of tools helps teams to organize their tasks, collaborate while developing code, enforce quality assurance, and facilitate applications build and deployments. Azure Boards, Azure Repos, Azure Pipelines, Azure Test Plans and Azure Artifacts are the default standalone services of Azure DevOps. The organizations in the Microsoft ecosystem frequently use these tools to create collaborative cultures and procedures that unite developers, project managers, and contributors. It allows companies to develop products and implement changes faster compared to traditional software development methods. Azure DevOps offers both an

## Chapter 3: Methodology

on-premises and a cloud solution called Azure DevOps Service [56]. Because it requires less setup time and resources, a cloud-based solution was chosen for this study. The study focuses on integrating security into software delivery automation within the context of DevOps. The image below shows the various features and services offered by Azure DevOps.



**Figure 3.3: Azure DevOps Services and its software development flow [57]**

The above image illustrates different services that could be used in different stages of SDLC. Azure DevOps flow is considered a continuous process which starts with planning and ends with monitoring. Project planning is the initial phase of software development. In this stage, project scopes are defined, user stories and relevant tasks are created, and those user stories and tasks are prioritized. In the Microsoft ecosystem Azure board service is used to facilitate those tasks. After that, developers will start software development based on the priorities of user stories and tasks. At this stage, source code collaborations are facilitated using Azure Repos service. As a third step, whenever the new feature development is complete, the source code is built and tested using the CI pipeline. After the successful completion, CI pipeline executable files are generated, and those executables are stored in Azure Artifacts. Then CD pipeline automatically fetches those executable files for the application deployment into various environments. Azure Pipeline service is used for the implementation of the CI/CD pipeline. Even though it is the final stage of the flow, but development team continuously monitors production applications to ensure application integrity and security. Azure Test Plans service could be used for extensive manual and automatic testing of applications [56].



## Chapter 3: Methodology

---

A project was created in an Azure DevOps organization to use the services that are required for the security automation test project. This project was implemented considering the software developer's perspective. Due to this, only Azure Repos, Azure Pipelines and Azure Artifacts are covered in detail. The implementation focuses on the development of completely automated continuous integration and continuous deployment process.

### 3.3.1 Azure Boards

Software development teams commonly use Azure Board as it provides interactive and scalable tools for managing software projects. It offers a wide range of features such as integrated reporting, calendar views and customisable dashboards. It also provides native support for Agile, scrum, and Kanban processes that facilitate software development. These tools have a track record of being quick and simple to use while tracking work progress, software-related concerns, and bugs. Azure Boards consists of a large variety of useful tools for different purposes like Work Items, Boards, Backlogs, Sprints, Queries and Delivery Plans [58]. Azure Boards is the service which facilitates the management perspective of DevOps, which is out of the scope of this study.

### 3.3.2 Azure Artifacts

Azure Artifacts are used to share different packages from both public and private sources within the team. Azure Artifacts can also be used to download and publish different kinds of packages to public registries and Artifacts feeds. Azure Artifacts and Azure Pipelines are used together for deploying packages from Azure Artifacts and publishing build packages to Azure Artifacts. Similarly, these packages can also be consolidated in build, test, or deployment stages in pipelines. Azure Artifacts supports build artifacts, NuGet, npm, Maven, PyPI, universal packages etc. [59]. For this research, Azure Artifacts were used to store the executables packages created during the build procedure.

### 3.3.3 Azure Test Plans

Azure Test Plans is a collection of tools that can be utilized to guarantee quality and cooperation throughout the development process. It is a browser-based solution which is used for test management. It is used for user acceptance testing, exploratory testing, manual testing, and gathering stakeholder feedback. The main benefits of Azure Test Plans are the ability to perform cross-platform tests, rich diagnostic data collection, end-to-end traceability, integrated analytics, and an extensible platform [60]. Software Quality Assurance was out of the scope of this study. Hence, it has been skipped in this study.

### 3.3.4 Azure Repos

Azure Repos is a version control system used for source code management. Like any other version control system, it is also widely used to save the snapshot of source code and track the changes. Those snapshots and changes are recorded in such a way that we can review and roll back to any version of the source code at any time. It is also essential for source code management and collaboration of code changes across the team [61]. Azure DevOps does not allow us to create a public repository in free trial and student subscription. At the same time, Snyk free version does not support private repositories. Therefore, Git repository was used for this study.

### 3.3.5 Azure Pipelines

Azure Pipelines are composed of one or more stages. Each stage is made up of one or more jobs, and each job contains one or more steps. Typically, pipelines are run in response to the triggers. Normally triggers are set up according to our business need to start the pipeline executions. It is a common practice to configure pipeline executions after new changes are merged in source control or at scheduled time intervals, or after a successful build process. Stages are used to organize multiple jobs in the pipelines, where each stage can have one or multiple jobs. We could have a separate stage for individual steps for example build, test, and deploy stages. We could run a job in an agent machine, and we could also have an agentless job. Steps are the smallest building block of a pipeline and can be a task or script. Tasks are specially designed scripts

## Chapter 3: Methodology

---

which perform a specific action, e.g., publishing a build artifact or making an API call. Some tasks might require running on the agent machine, and some can be agentless [62]. The agent is a cloud computing infrastructure with agent software installed that executes one job at a time. We need at least one agent to build or deploy our software using Azure pipelines. More agents may be needed depending on the number of executables to be built from the code, the software to be deployed, and the required numbers of users access.

Azure Pipelines provides build and release services that aid in continuous integration (CI) and continuous delivery (CD) in SDLC. Azure Pipelines is platform-independent and supports almost any programming language or project type. The DevOps team creates a build pipeline configuring tasks to test, build and publish artifacts. Continuous integration is enabled in the build pipeline to automatically trigger its tasks whenever the new changes are merged into the version control. Automated tests are also executed in CI processes to ensure quality. Automated tests are effective in catching the bugs early in the development process resulting in bug fixes being cost-effective. Azure builds pipelines, builds artifacts and publishes them. The published artifacts are used by the release pipelines for application deployment to development, QA, staging and production environments [63]. The release process is automated by setting a trigger that runs the release pipelines after the CI process is executed successfully. Azure Pipelines was used to develop a fully automated DevOps CI/CD pipeline, and the details of it will be explained in the implementations segment.

### 3.4 Security Integration in Azure DevOps

DevSecOps has always been a vague concept that starts from building secure work culture and incorporating best security practices starting from design, development, quality assurance, security audits and continuous system monitoring, which is a challenging task even for big software corporations. Security guidelines and best practices require continuous updates to mitigate continuously emerging vulnerabilities. To address these challenges, there is a need for security scanning tools that could be integrated into SDLC, which would empower developers to enforce security. Integrating automated security scanning tools in DevOps practices has not yet become

## Chapter 3: Methodology

---

widespread. The security community has not yet recommended any specific set of tools that can be considered reliable.

Numerous security scanning tools are currently available for scanning security vulnerabilities. As previously mentioned, most of these tools fall under various categories, including SAST, DAST, vulnerable dependency scanners, secret scanners, container vulnerability scanners, IaC vulnerability scanners, and cloud vulnerability scanners. To incorporate security into the early stages of development, it is important to have a dependable tool that can be used in the IDE to enable developers to enforce security while developing an application. However, most of the available tools are limited to automating security tests on source control systems and CI pipelines. Although there are some tools that can be used throughout the SDLC, they are mostly paid versions. For this study, an open-source tool called Snyk was used to integrate security scanning into a completely automated DevOps CI/CD pipeline. More information about its implementation will be provided in the upcoming sections.

### 3.4.1 Snyk

The Snyk platform has the capability to identify and resolve vulnerabilities in various areas, including custom code, open-source dependencies, container images, and IaC configurations. It can be integrated into different stages of software development, such as IDE, SCM systems, and DevOps automation pipelines [64]. The platform offers several tools like Snyk Code, Snyk Open Source, Snyk PR Checks, Snyk Container, Snyk IaC, and Snyk Cloud. These tools were created to address application security, software supply chain security, and cloud security concerns. They are designed to protect code, containers, and deployment. Snyk offers several products, including Snyk Web UI, Snyk CLI, Snyk IDE extension, and Snyk API, which can be used to perform security scans [65]. Snyk products provide features such as Snyk Open Source and Snyk Code, which can perform Static Application Security Testing (SAST) and Software Composition Analysis (SCA). These products have various integrations, including IDE, git repository, and CI/CD integrations, which were implemented in this study [66]. The study focused on Snyk CLI, Snyk IDE, Snyk PR Checks, and Snyk IaC while leaving other Snyk tools for future investigations.

## Chapter 3: Methodology

---

### 3.4.1.1 Snyk Open Source

Like any other SCA tool, Snyk Open-Source scans all the dependencies that exist in the application and checks if it contains any dependencies that have been marked as vulnerable in Snyk Vulnerability Database. The Snyk Vulnerability Database is similar to NVD and is a customized database of vulnerabilities. It is continually updated to stay synchronized with NVD updates. The Snyk open-source tool is utilized to detect vulnerabilities in open-source libraries. Whenever a vulnerable library is detected, the tool produces a report detailing the vulnerability and provides recommendations to address the issue [67]. Snyk Open-Source examines the language and package manager of the project and generates a dependency tree accordingly. In the .NET ecosystem, the dependencies are called NuGet packages. Once the project dependencies are restored, the framework generates a project.assets.json file using the \*.proj, \*.proj \*.csproj, \*.vbproj, and \*.fsproj files. Snyk Open-Source security scanning can only be performed after creating a project.assets.json file [68].

### 3.4.1.2 Snyk Code

Snyk Code is similar to other SAST tools and is used to identify vulnerabilities in source code, bytecode, or binary code. Typically, SAST tools are specific to a particular programming language [69]. The Snyk company asserts that Snyk Code is consistently broadening its knowledge base regarding code security due to its integration with a semantic analysis AI engine that gains insights from millions of open-source commits. Additionally, it is combined with the Snyk Security Intelligence database. Snyk Code scanning does not need projects to be compiled, and it can scan source code as it is being written. Security scans are fast and produce a detailed and useful report on any vulnerabilities found. The report contains detailed information on the vulnerability along with its CVE score, and it also recommends the fix along with example code [70]. Snyk uses an AI-based analysis engine to detect hardcoded secrets, coding issues, type inference, value ranges, data flow, API usage, control flow and point-to-analysis [71].

### 3.4.2 GitGuardian

GitGuardian is a program that scans source code in real-time for API keys, passwords, certificates, encryption keys, and other types of sensitive information. It uses an automated detection engine to detect secrets in the SDLC. Developers can use this tool to scan both public and private code repositories and will send alerts when secrets are exposed in the repositories, thereby reducing the risk of secret exposure. In this study, it was used to ensure that secrets were checked before merging the pull request into the master [72].

# Chapter 4

## 4 Implementation

### 4.1 Test Projects

Test projects are crucial for the evaluation of the available security scanning tools based on their performance. At the same time, it is challenging to figure out the right test cases that might produce valuable results. Since the proposed research topic had a fair amount of research and implementational complexity, this study was conducted to cover a lot of ground while leaving enough room for further development and research. Keeping this in mind, the whole implementation and research was conducted to cover the most important security concerns in this study. In this section, three test projects were implemented to address specific purposes. The first project was developed to test the security challenges of the old production applications infrastructure. The second part is infrastructure as a code implementation for managing and provisioning the application infrastructure using the code instead of a manual process. The last one implemented was OWASP Web API top 8 vulnerabilities so that it could be used to test the performance of the available Static Application Testing tool.

#### 4.1.1 Production Test Application

A sample web API application and a console application was developed using asp.net core framework version 3.1, released in 2019. Similarly, all the libraries and packages were also selected from 2019 so that we will have a better understanding of new vulnerabilities that appeared after 2019. Since the actual project implementation and coding were out of the scope of this study. Due to that reason, the actual code of the project has not been discussed.

## Chapter 4: Implementation

### 4.1.1.1 Web API

A web API was created with certain features, including an asynchronous endpoint that can receive and process HTTP post requests. The endpoint first checks whether the received data is valid and processes only the valid ones. The controller calls BlobService to log the event in a storage account. In order to do so, the WriteLog method is used to verify if there is a user-specific daily blob file. If the file exists, the events are logged into that specific file. If there is no existing blob log file, the system creates a new one and starts logging the events. The email addresses are hashed for privacy while logging the events. All the events of the specific user are saved in the user's blob file. Following the logging process in the previous step, the MessageQueueService is invoked to transmit the messages to the storage account queue for further processing. To ensure the data is transmitted accurately, the payload data is converted to base64 encoding. The transmission of the message to the Azure storage queue is done asynchronously. All significant steps in the entire process are recorded in the same blob log file created in the previous step, which is specific to the user. The implementation of this process can be demonstrated by the code map given below.

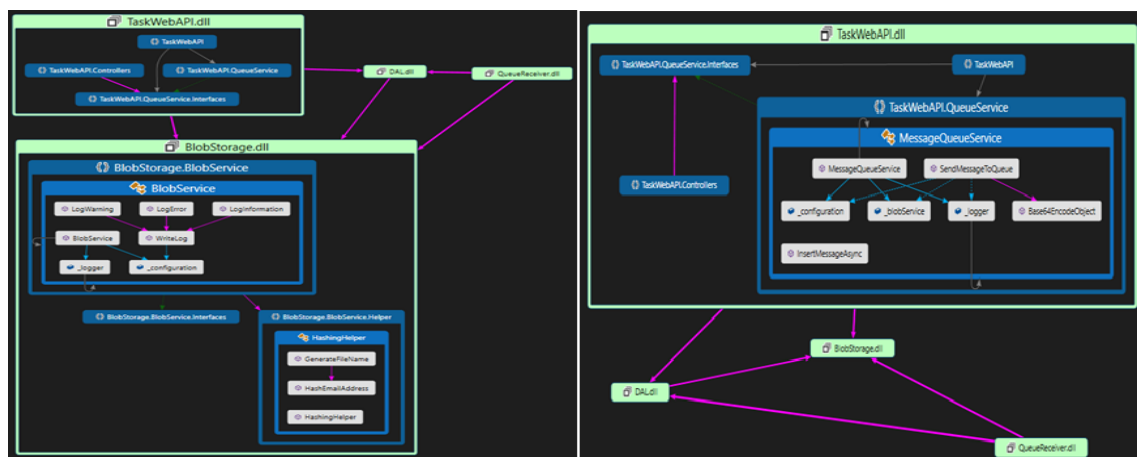


Figure 4.1: Code Map of event log service and queue service implementation

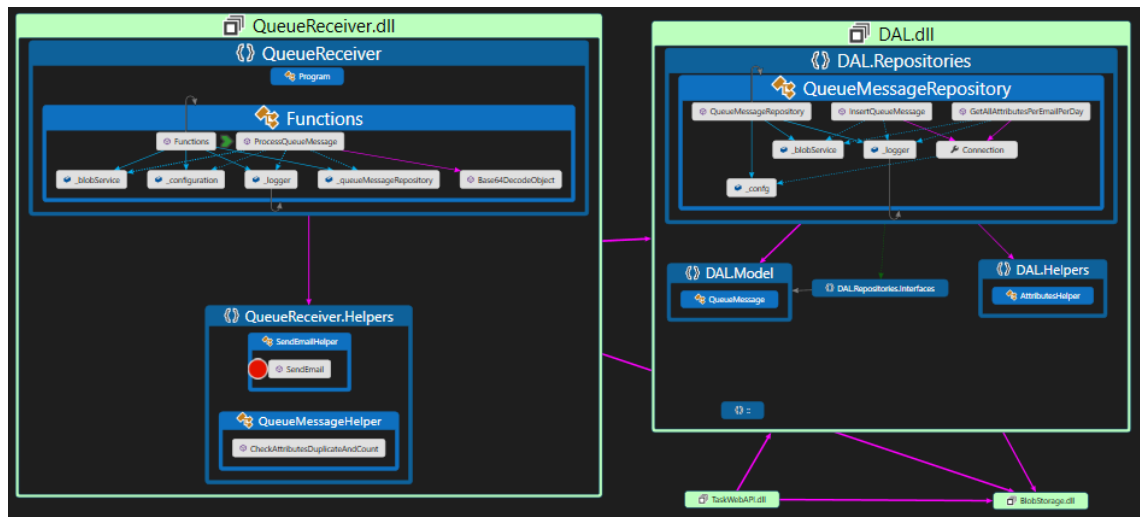
### 4.1.1.2 Console Application

A second microservice was created in the form of a console application named QueueReceiver to receive messages that were stored in the storage account queue by



## Chapter 4: Implementation

MessageQueueService. The Base64 encoded messages were decoded and converted to a message object. Then, the existing message attributes for that user were extracted from the SQL message table and compared with the received message attributes to identify any duplicate attributes. The number of existing attributes was also counted. Only the new messages without duplicate attributes were processed and stored in the SQL message table. If the count of message attributes was equal to or greater than 10, a congratulatory email with the list of attributes was sent to the user using the third-party service integration of Trillo SendGrid API for reliable email delivery. All the significant events were written to the same individual user-specific daily blob file using the console application, as in the previous steps. The implementational flow of this process is illustrated in the image below.



**Figure 4.2: Code Map of Queue Receiver**

Azure app service was selected to host both web API and a console application. The console application was deployed as a WebJob. Within this context, a storage account was utilized to store various data objects, such as blobs and queues. To store structured data securely, SQL server was employed. For cloud-based identity and access management services, Azure Active Directory (Azure AD) was utilized [73]. All the sensitive information, like database connection string, storage account connection string, SendGrid API key etc., were stored in Azure Key Vault. Azure AD's managed identity service was used to enforce a coherent, unified access policy for hosted applications (i.e., Web API and Web job) and developers. Using this feature application and user may easily access other resources protected by Azure AD, such as Azure Key

## Chapter 4: Implementation

---

Vault, without any additional configurations in the application code and development machines [74].

### 4.1.2 Infrastructure as Code Implementation

The basic structure of an ARM (Azure Resource Manager) template includes several elements such as schema, contentVersion, parameters, functions, variables, resources, and outputs, which are essential for implementing Infrastructure as Code (IaC). The schema element is mandatory, and it specifies the location of the schema file and language version that will be used as required. Similarly, the contentVersion element is also necessary and contains a version value for the deployment file's versioning. The parameter element is optional, but if included, it contains custom values for deploying resources. Functions are also optional and define different functions available within the template. The variables element is optional and, if included, contains values that define JSON fragments that simplify the template. The resources element is a key required element that contains resource types for deployment during template execution. Finally, the outputs element is optional and defines the values returned after successful resource deployment [75]. In this study, functions and variables were out of the scope, so they are left for further improvements.

#### 4.1.2.1 Parameters

Normally all the required parameters are defined with their respective values. These values are normally resolved by the Resource Manager before starting the deployment operation. Resource Manager replaces the parameter with the resolved value. ARM template can only have 256 parameters. When defining the parameters, we must have a name and value attributes. Other available optional attributes are secure parameters, allowed values, default values, length constraints, Integer constraints and description. Parameters can be used to pass resource names, app configuration references and key vault references. Parameter functions can be used to reference parameters in the template. ARM templates are normally reused to deploy the resources into different environments. Due to that reason, environment-specific parameter files are used to define the environment-specific values. Those environment-specific parameter template files are used while deploying different resources in various environments. To

## Chapter 4: Implementation

---

accomplish that, ARM templates normally only contain the parameter name and type. But the actual values are passed through parameter files [76]. In the following code snippet, we can see an example parameter that has been used in this implementation.

```
"parameters": {
  "serverFarmsName": {
    "type": "String"
  },
  "appServiceName": {
    "type": "String"
  },
  "sqlSarverName": {
    "type": "string"
  },
  "sqlDBName": {
    "type": "string"
  },
  "sql-cs-key": {
    "type": "string"
  }
}
```

### Code Snippet 4.1: Template parameters used for the implementation.

The parameter files, like ARM template files, include the location of the file and versioning information. The parameter values required for resource deployment are then assigned in the parameter element. For different environments, such as development, quality assurance, and production, separate parameter files are usually created. Because the parameter file stores parameter values in plain text, it cannot be used to store secret information. Therefore, all the secret values are stored in the key vault, and their references are used in the parameters file [77]. The following code snippet shows some of the parameters that have been used in the study.

## Chapter 4: Implementation

---

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "serverFarmsName": {
      "value": "SecurityTestAutomationSF-prod"
    },
    "appServiceName": {
      "value": "SecurityTestAutomationAS-prod"
    },
    "sqlSarverName": {
      "value": "SecurityTestAutomationSQLServer-prod"
    },
    "sql-cs-key": {
      "value": "ConnectionStrings--ProdDB"
    }
  }
}
```

### Code Snippet 4.2: Production environment parameters in the parameters file.

#### 4.1.2.2 Resources

The resources section in an ARM template is used to specify the resources that will be deployed when the template is run. A single template can only contain up to 800 resources. Within the resource element, there are several sub-elements, such as type, apiVersion, name, comments, location, dependsOn, sku, identity, kind, scope, properties, and resources. The type element specifies the resource type, which includes the resource provider namespace and the resource type itself. The apiVersion element specifies the version of the REST API used to deploy the resource. The name element specifies the name of the resource. The location element specifies where the resource should be deployed. The dependsOn element is used to indicate that a resource depends on another resource and cannot be deployed until the dependency is met. The sku element specifies the service tiers that will be used when deploying the resource. If a resource supports managed identity, the identity element can be used to define its identity. Any sub-resources are defined within the resource sub-element and will be deployed alongside the main resource [75]. The following code snippet illustrates the resource declaration to deploy the app service, which hosts web API and console application discussed in the previous section.

## Chapter 4: Implementation

---

```
"resources": [
  {
    "type": "Microsoft.Web/serverfarms",
    "apiVersion": "2021-03-01",
    "name": "[parameters('serverFarmsName')]",
    "location": "East US",
    "sku": {
      "name": "F1"
    },
    "properties": {}
  },
  {
    "type": "Microsoft.Web/sites",
    "apiVersion": "2021-03-01",
    "name": "[parameters('appServiceName')]",
    "location": "East US",
    "dependsOn": [
      "[resourceId('Microsoft.Web/serverfarms', parameters('serverFarmsName'))]"
    ],
    "identity": {
      "type": "SystemAssigned"
    },
    "properties": {
      "serverFarmId": "[resourceId('Microsoft.Web/serverfarms',
parameters('serverFarmsName'))]"
    },
    "resources": [
      {
        "type": "Microsoft.Web/sites/config",
        "apiVersion": "2021-03-01",
        "name": "[concat(parameters('appServiceName'), '/appsettings')]",
        "kind": "string",
        "properties": {
          "VaultUri": "[parameters('vaultUri')]",
          "AzureWebJobsDashboard": "[parameters('azureWebJobsDashboard')]"
        },
        "dependsOn": [
          "[resourceId('Microsoft.Web/sites', parameters('appServiceName'))]"
        ]
      }
    ]
  }
]
```

**Code Snippet 4.3: Code for the deployment of the service app**

## Chapter 4: Implementation

---

The above code snippet uses the parameters function to retrieve the values from the parameters file while the resource is being deployed. To host app service, we need server farms, websites and app configurations as a resource for deployment. In the App Service resource, it is marked that it depends on a server farm. Similarly, in the App Config resource, it is marked it depends on App Service. Due to that reason, when Resource Manager deploys Server Farm first, then App Service, and at last, it deploys App Config. Since App Service supports managed identity, the system assigned identity is used so that App Service can communicate with other Azure resources using system assigned identity. East US location was used because all the free service tiers were only available in that location. App Configuration properties element is used to store application-specific configurations. A parameter file has been used to retrieve parameter values to be used in the resources. For this application to work, we also need Key Vault, Storage Account and SQL server. Storage Account contains blob service, queue services, containers, and queue resources. Similarly, SQL servers contain databases. Similar scripts should be written for the creation of Storage Account and SQL server resources.

### 4.1.2.3 Outputs

The output section is used to define the values which are returned after the successful deployment of the resources. It is used to keep track of the process of deployment as well as to check if the deployment has been completed as expected. Currently, we can only use 64 outputs in a template. The output section contains sub-elements like output name, condition, type, value, and copy. For the resource that has been defined earlier following output values in the code snippet were defined.

## Chapter 4: Implementation

---

```
"outputs": {
  "serverFarmsName": {
    "type": "string",
    "value": "[parameters('serverFarmsName')]"
  },
  "serverFarmsId": {
    "type": "string",
    "value": "[resourceId('Microsoft.Web/serverfarms', parameters('serverFarmsName'))]"
  },
  "appServiceName": {
    "type": "string",
    "value": "[parameters('appServiceName')]"
  },
  "appServiceId": {
    "type": "string",
    "value": "[resourceId('Microsoft.Web/sites', parameters('appServiceName'))]"
  },
}
```

### Code Snippet 4.4: Server Farm and App Service outputs declarations

#### 4.1.3 Vulnerable Test Project

OWASP Top 10 API Security risk published in 2019 was used to implement the vulnerable Web API [35]. This vulnerable API was developed to test the performance of available security scanning tools.

##### 4.1.3.1 Broken Object Level Authorization Vulnerability

Broken Object Level Authorization could cause information disclosure to unauthorized users. Attackers could easily take advantage of this vulnerability to have unauthorized access to sensitive data and illegally modify or delete those data. The following endpoint was implemented as an example of broken object-level authorization. Even though the code seems to have proper checks for the user roles and the ownership of the resources, it still contains that vulnerability because it lacks to ensure that the user has the appropriate permission to access the specified resource. It would be better to check the user's permissions to access the specific resource rather than just checking the user's role. This issue could be solved using claims-based authorization using ClaimsPrincipal

## Chapter 4: Implementation

---

or `AuthorizeAttribute` which gives us more control to fine-tune access control for the resources.

```
[Authorize]
[HttpGet]
public IActionResult GetResource(int id)
{
    var claimsIdentity = (ClaimsIdentity)User.Identity;
    var userName = claimsIdentity.Name;
    var userRole = claimsIdentity.FindFirst(ClaimTypes.Role).Value;

    var resource = _resourceRepository.GetResource(id);
    if (userRole == "admin")
    {
        // Allow access
        return Ok(resource);
    }
    else if (userRole == "user" && resource.Owner == userName)
    {
        // Allow access
        return Ok(resource);
    }
    else
    {
        // Deny access
        return Forbid();
    }
}
```

### Code Snippet 4.5: Broken Object Level Authorization Example

#### 4.1.3.2 Broken User Authentication Vulnerability

Broken user authentication could allow attackers to compromise authentication tokens and impersonate an authentic user identity for a short period or permanently. The following login endpoint was implemented as a broken user authentication example. The Endpoint only checks username and password. But it does not check another key issue, like if the user is locked out or the password is expired. Similarly, the password is also stored as plain text in the database. After the successful login, the endpoint returns a token which will be later used to authorize access to the protected resources. Token generation logic in the example is insecure because it does not use encryption and signing mechanism. A standard token generator must use JWT tokens which contain a symmetric security key and secured algorithm to sign the token, claims, issuer, audience, and expiry information.



## Chapter 4: Implementation

---

```
[HttpPost("Login")]
public IActionResult Login([FromBody] LoginModel model)
{
    var user = _userRepository.GetUser(model.Username, model.Password);

    if (user != null)
    {
        var token = _encryptionService.GenerateToken(user);
        return Ok(new { token });
    }
    else
    {
        return Unauthorized();
    }
}

public string GenerateToken(User user)
{
    var token = user.Username + ":" + DateTime.Now.ToString();

    return Convert.ToBase64String(Encoding.ASCII.GetBytes(token));
}
```

### Code Snippet 4.6: Broken User Authentication Example

#### 4.1.3.3 Excessive Data Exposure Vulnerability

Excessive Data Exposer could be easily exploited by the attacker by sniffing the network traffic and analysing the endpoint responses. The following endpoint exposes a complete user object which contains all sorts of sensitive information like SSN, password, credit card number, bank account number etc. In this situation, an attacker could easily obtain this PII information and conduct various serious attacks. This problem could be solved by a simple solution where we could cherry-pick the required parameters like Id, username, first name and last name to create a DTO for the user objects which does not contain any sensitive information. Then we could return that user DTO object as a response to the request.

## Chapter 4: Implementation

---

```
[HttpGet]
public IActionResult GetUser(int id)
{
    var user = _userRepository.GetUser(id);
    return Ok(user);
}

public class User
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Username { get; set; }
    public string Salt { get; set; }
    public string HashPassword { get; set; }
    public string Role { get; set; }
    public string SSN { get; set; }
    public string CreditCardNumber { get; set; }
    public string BankAccountNumber { get; set; }
}
```

### Code Snippet 4.7: Excessive Data Exposer Example

#### 4.1.3.4 Lack of Resource and Rate Limiting Vulnerability

Lack of resources and rate limiting restriction can cause API endpoint to be vulnerable to denial of service (DoS) attacks and brute force attacks. According to OWASP, it is quite common in the existing API's endpoints. The following endpoint could be considered as an example that does not limit resource and request rate. It does not restrict file types and file sizes. To orchestrate the heavy processing load situation a method with large numbers of iterations was used. It could consume large amounts of CPU and memory resources. Due to all these reasons, it could be considered a good example endpoint that lacks resources and a rate limit. To secure this endpoint, we could use the .net core "AspNetCore.RateLimit" library to implement restrictions on the number of requests that a specific client can make within the given time. We could also restrict the input file to be a specific type and also restrict the file size in this context.

## Chapter 4: Implementation

---

```
[HttpPost]
public async Task<IActionResult> ProcessFile(IFormFile file)
{
    if (file == null || file.Length == 0)
    {
        return BadRequest("File not found");
    }
    _logger.LogInformation("File received");
    var result = await _heavyProcessingService.ConductHeavyProcessing();
    return Ok(result);
}

public async Task<int> ConductHeavyProcessing()
{
    _logger.LogInformation("Heavy Processing method called!");
    int result = 0;
    for (int i = 0; i < 1000000000; i++)
    {
        result += i;
    }
    return result;
}
```

### Code Snippet 4.8: Endpoint that Lacks Resource and Rate Limit Vulnerability

#### 4.1.3.5 Broken Function Level Authorization Vulnerability

Broken functional level authorization could be exploited by the attacker to gain illegal access to sensitive users' resources and administrative functions. Even though it is a complex problem and usually has a complex hierarchy of groups and roles. But a simple endpoint was used since the scope of this study was to test the performance of the SAST tool rather than studying the complex hierarchy of access management. This example does not have any function-level authorization, which means anyone could access this administrative endpoint. Since it is an administrative function, it should have authorization for the user in an admin role.

```
[HttpGet]
public IActionResult GetSensitiveData()
{
    var data = _userRepository.GetAllUsers();
    return Ok(data);
}
```

### Code Snippet 4.9: Broken Functional Level Authorization endpoint.

## Chapter 4: Implementation

---

### 4.1.3.6 Mass Assignment Vulnerability

Mass assignment is quite common in APIs because they usually expose the application business logic and properties names. Due to the automatic request body binding feature of modern frameworks, developers are compelled to make this mistake. Attackers could use this existing mechanism for privilege escalation, data tempering and security bypass. The following example uses the automatic binding of the request body with the sensitive user object. This example code does not use parameter whitelisting or blacklisting mechanisms to restrict administrative parameter assignment. Due to that reason, it is vulnerable to mass assignment. To overcome these issues, we could create a user DTO object with permitted parameters to secure administrative parameters which should not be accessed by the user.

```
[HttpPost("UpdateUser")]
public IActionResult UpdateUser(User user)
{
    _logger.LogInformation($"User {user}");
    user.Salt = Guid.NewGuid().ToString("N");
    user.HashPassword = _encryptionService.HashPassword(user.Password, user.Salt);
    var result = _userRepository.UpdateUser(user);
    return Ok("User Updated Successfully!");
}
```

#### Code Snippet 4.10: Endpoint with Mass Assignment flaw

### 4.1.3.7 Security Misconfiguration Vulnerability

Attackers could try to find unpatched security flaws, generic sensitive endpoints, and unprotected resources to gain unauthorised access. The following code was used to test the performance of the SAST tool. It clearly illustrates that the application exceptions are exposed in the exception page, anonymous clients are allowed to make any kind of HTTP request, unsecured HTTP requests are not redirected to secured HTTPS, and authorized users have full access. These vulnerabilities could be fixed with proper error handling with quarantined error response schema, enforcing CROS policy allowing access to the specific clients to the specific HTTP method according to the requirements and redirecting all unsecured HTTP requests to secured HTTPS endpoints.

## Chapter 4: Implementation

---

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseDeveloperExceptionPage();

    app.UseCors(options => options.AllowAnyOrigin().AllowAnyMethod());

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

### Code Snippet 4.11: Studied Security Misconfiguration

#### 4.1.3.8 Injection Vulnerability

An attacker could use this vulnerability by sending crafted input data to the endpoint to access sensitive PII, grant admin rights or manipulate or destroy the sensitive data. The following code indicates that the received request is directly mapped with the category object without any modal validation logic. Model validation attributes could have been used in the model class to enforce model validation. Similarly, the received parameters are directly concatenated in the SQL query such that they will be directly passed to the database interpreter to execute the query. In this situation, the database is wide open for all sorts of unauthorized SQL query execution. It could have been prevented by enforcing model validation before binding the data to the category model. Category class could have used model validation attributes to facilitate model validation. Parameterized SQL query should have been used instead of the concatenated query so that the crafted SQL query in the parameters would have been prevented from execution by the database interpreter.

## Chapter 4: Implementation

```
[HttpPost]
public IActionResult AddCategory(Category category)
{
    var result = _categoryRepository.InsertCategory(category);
    if(result > 0)
    {
        return Ok("Category Successfully Added!");
    }
}

public class Category
{
    public int Id { get; set; }
    public string CategoryName { get; set; }
    public string Description { get; set; }
}

public int InsertCategory(Category category)
{
    string queryString = $"insert into Category (CategoryName, Description) Values
('{category.CategoryName}', '{category.Description}')";
    using(SqlConnection connection = (SqlConnection)Connection)
    {
        var result = connection.Execute(queryString);
        return result;
    }
}
```

Code Snippet 4.12: SQL Injection Example

## 4.2 Azure DevOps and CI/CD Pipeline

Ideally, in all organizations, development flow starts only after completing project planning. After planning, the project backlog is populated, creating well-defined work items along with project requirements. Then in the next phase, DevOps teams create CI/CD pipelines and cloud infrastructure according to the requirements. After that, the actual software implementation phase starts, and the developer starts implementing the feature in the work item. After the feature is fully implemented and thoroughly tested, the source code is merged into the version control system using a pull request. This action then activates the build pipeline, which builds and publishes the artifacts. The following image showcases the initial development flow.

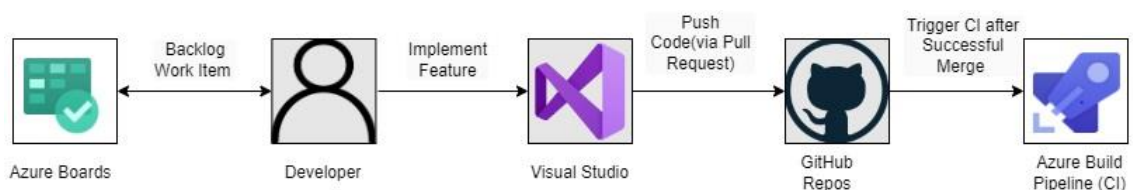


Figure 4.3: Initial Development Flow

## Chapter 4: Implementation

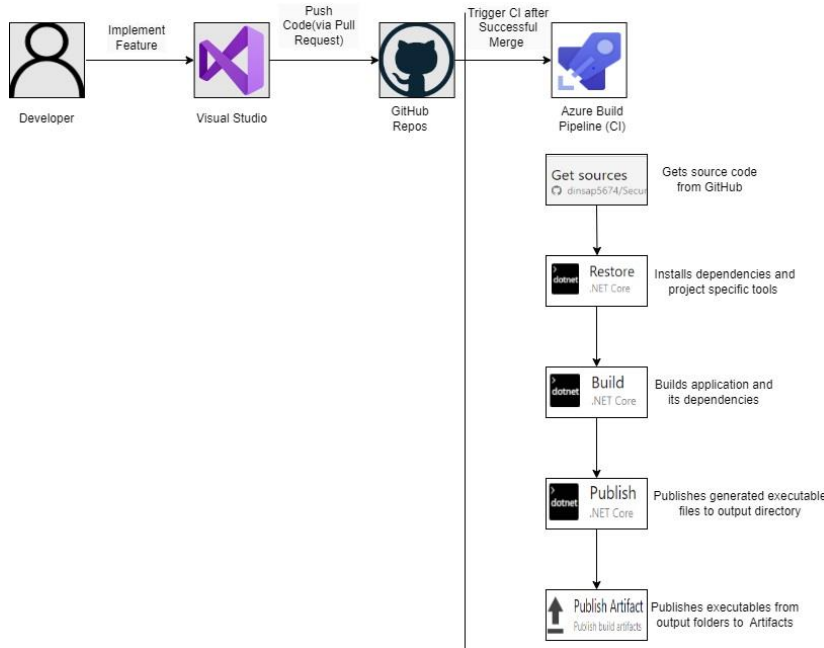
---

In the above flow, we observe that the developer selects a user story and undertakes various tasks associated with it. Once the implementation is finalized, the developer proceeds to push the changes to the git repository via a pull request. Following the successful merging of the changes into the master branch, the SCM system triggers the build pipeline. It is worth mentioning that Azure DevOps supports integration with GitHub, and in this implementation, GitHub has been used due to the previously mentioned reason.

### 4.2.1 Continuous Integration using Azure Build Pipelines

Azure build pipeline can be considered as a building job that executes different tasks in the specified sequence. A simple build pipeline contains different tasks for getting source code from SCM system, installing the required tools (i.e., NuGet Packages and 3<sup>rd</sup> parties' libraries), building solutions, generating executable packages, running tests, and finally publishing those packages to artifacts. It is similar to the tasks that are performed locally in the local machine when a project is built and executed. In the conventional deployment process, release configurations were used to generate executable packages, which were subsequently manually deployed by developers to the production environment. Similarly, in the context of a build pipeline scenario, a comparable Azure machine is required to execute all of those identical tasks. Azure Build Pipelines utilize agent machines to carry out these tasks. The agent machine can either be a machine provided by Azure and rented for our usage, or it can be a self-hosted machine located on-premises or in the Azure virtual environment. While the build pipeline offers a wide range of testing capabilities, they are not included in this particular implementation. The following flow diagram shows the various tasks and their sequential execution within the build pipeline.

## Chapter 4: Implementation



**Figure 4.4: Complete Flow Until Build Pipeline**

In the above flow diagram, whenever a SCM system triggers-built pipeline, as a first step, it extracts the latest code from the master branch. Then the pipeline installs project-specific tools and dependencies as the second step. After the restore process is completed, it triggers the built process to build the artifacts. In the publishing step, those executable files are published to the output directory. As a final step, those executables are extracted from the output folder and stored in Azure artifacts, which could be used by release pipelines.

When we create a build pipeline, we need to select the source code repository that the pipeline uses to get the source code. But this implementation uses a third-party SCM system as we need to create a connection from Azure Pipelines to GitHub. In Azure DevOps, we can create service connections to grant access to the Azure Pipelines to the external services. Only after that build pipeline will be able to check out the source code before the execution of the build task [78]. After that CI can be enabled in the SCM system. Then whenever a new code is merged into the repository, it triggers the build process defined in the CI pipeline. It is also possible to enable triggers in different git branches. But for this context, the trigger is set only in the master branch.

In this implementation, the Azure-hosted agent pool was chosen as it provides preconfigured host machines that are readily available for use. The host machine within



## Chapter 4: Implementation

---

the agent pool was configured with the latest Windows operating system to facilitate the building of executable artifacts [79]. YAML files are commonly employed for configuring CI/CD pipelines. YAML, a human-readable data serialization format, is frequently utilized for creating configuration files and facilitating data exchange between various programming languages. Similarly, the YAML configuration file can be utilized to define workflows for building, testing, and deploying within the CI pipeline [80]. The provided code snippet shows the configuration of the host for the CI pipeline.

```
trigger:
  - master

pool:
  vmImage: 'windows-latest'

variables:
  BuildConfiguration: 'Release'
```

### Code Snippet 4.13: Trigger, agent and build configuration.

In the code example above, the configuration of the CI pipeline was set to activate whenever there are new merges in the GitHub master branch. Additionally, an Azure-hosted agent was designated, utilizing the most recent Windows operating system for building executable files. The build process was further configured to generate a release directory dedicated to storing the resulting executable files.

Once these configurations are in place, the next step is to define the various tasks that need to be executed during the build process or build job. In this particular implementation, the hosted agent utilizes .NET Core CLI to carry out tasks such as restoring dependencies, building the project, and publishing the output. The pipeline process initiates the restore task, which uses NuGet to install all the required dependencies and project-specific tools specified in the project file (e.g., .csproj file). Subsequently, the build task is performed, resulting in the creation of the application and its dependencies in the form of executable binaries (e.g., .dll) and additional package files (e.g., .pdb, .deps.json, .runtimeconfig.json, etc.) [79]. The following code snippet shows those configurations in YAML file.

## Chapter 4: Implementation

---

```
steps:
- task: DotNetCoreCLI@2
  displayName: Restore
  inputs:
    command: 'restore'
    projects: '**/*.csproj'
    feedsToUse: 'select'

- task: DotNetCoreCLI@2
  displayName: Build
  inputs:
    command: 'build'
    projects: '**/*.csproj'
    arguments: '--configuration $(BuildConfiguration)'
```

### Code Snippet 4.14: Pipeline restores and build configurations.

Once the build task is successfully completed, the publish task is executed, which compiles both applications and verifies the dependencies specified in the project file. Subsequently, it publishes the generated files to the output directory called `ArtifactStagingDirectory`. However, this particular step poses a challenge due to our application architecture, as we are attempting to publish Web API and Console application from the same build pipeline [79]. In the publishing stage, we had three sub-tasks. The first two tasks utilized the .NET Core CLI to extract the executable files from the release directory. These extracted executables were then published to the publish-output directories in their respective locations, making them easily accessible for the CD pipeline to deploy to the Azure app service. The final task involved publishing artifacts from the output directory and archiving them in a designated location with specific directory structures, all in a compressed zip format. The necessary configurations for this process are illustrated in the code snippet below.

```
- task: DotNetCoreCLI@2
  displayName: Publish WebAPI
  inputs:
    command: 'publish'
    publishWebProjects: true
    arguments: '--configuration $(BuildConfiguration) --output
$(Build.BinariesDirectory)/publish_output'
    zipAfterPublish: false
    modifyOutputPath: false
```

## Chapter 4: Implementation

---

```
- task: DotNetCoreCLI@2
  displayName: Publish WebJob
  inputs:
    command: 'publish'
    publishWebProjects: false
    projects: '**/QueueReceiver.csproj'
    arguments: '--configuration $(BuildConfiguration) --output
$(Build.BinariesDirectory)/publish_output/App_Data/jobs/continuous/QueueReceiver'
    zipAfterPublish: false
    modifyOutputPath: false

- task: ArchiveFiles@2
  inputs:
    rootFolderOrFile: '$(Build.BinariesDirectory)/publish_output'
    includeRootFolder: false
    archiveType: 'zip'
    archiveFile: '$(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip'
    replaceExistingArchive: true
```

### Code Snippet 4.15: Publish configuration for the project implementation.

Finally, publish artifact task gets those achieved artifacts from the output directory and publishes them to the artifact called the drop [79]. The following configuration code defines the task as publish artifacts, specifies the existing artifact's location, and defines the artifact name, and publish location.

```
- task: PublishBuildArtifacts@1
  displayName: 'Publish Artifact'
  inputs:
    PathToPublish: '$(Build.ArtifactStagingDirectory)'
    ArtifactName: 'drop'
    publishLocation: 'Container'
```

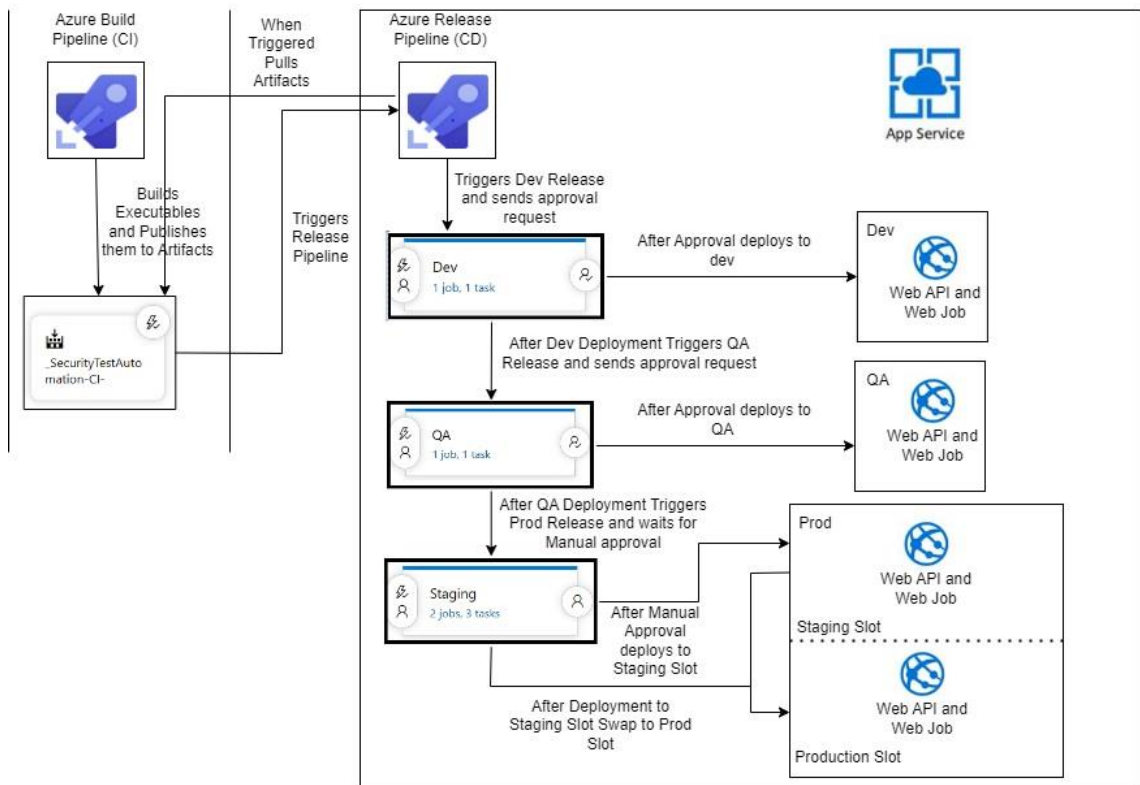
### Code Snippet 4.16: Publish artifacts tasks.

The build process typically proceeds to the next task only after successfully completing the current task. While the test task holds significant importance in the build pipeline for ensuring software quality, it is not within the scope of this study and is thus left for future development. This stage also serves as an appropriate point to conduct essential security testing and scanning before releasing the changes to the production environment. The integration of security scanning within the CI pipeline will be discussed in the section dedicated to security integration.

## Chapter 4: Implementation

### 4.2.2 Continuous Deployment using Azure Release Pipelines

The main aim of the project implementation was to develop fully automated continuous deployment pipelines. Azure classic pipelines were used to implement the required release pipeline. The implementation was developed, creating a new Azure release pipeline in Azure DevOps using the available Azure app service deployment template. The release pipeline is configured to deploy both web API and web job applications in the Azure app service. This release pipeline is set up to use the latest artifacts produced by the build pipeline in the previous step. A continuous deployment trigger is enabled to automate the execution of the release pipeline after the build pipeline successfully builds the new artifacts. The following diagram shows the complete flow for the executions of different tasks in the CD pipeline.



**Figure 4.5: Complete CD pipeline Execution Flow**

The above diagram illustrates the complete architecture of the CD pipeline that has been implemented to fully automate a complete continuous release process. As discussed earlier, whenever new executables files are published in the artifacts drop directory. That event triggers the release pipeline and extracts the latest application artifacts.

## Chapter 4: Implementation

---

Those artifacts are used by the release pipeline to deploy applications into the pre-defined deployment environments.

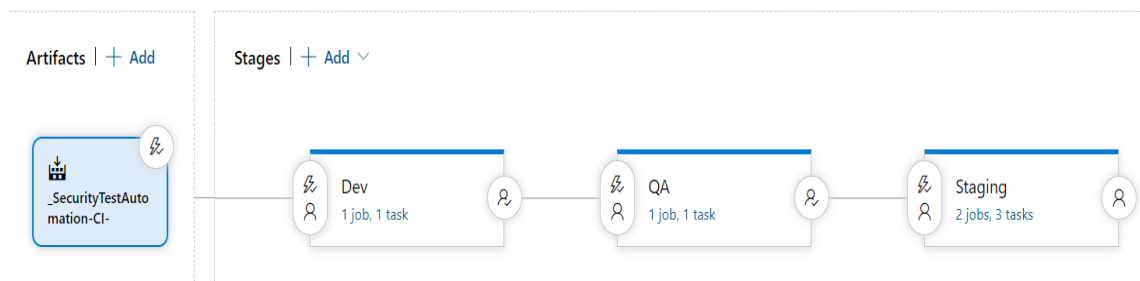
In this study, development, QA, and staging environment release were implemented in three different steps. After the successful extraction of the artifacts CD pipeline triggers a development release step. An approval policy has also been configured for all deployments as a safety net to prevent accidental deployment. Due to that reason, the pipeline sends approval requests, and after receiving the approval for the deployment, the pipeline starts the process for the deployment of the application to the development environment. After the successful deployment of the application, the pipeline triggers the QA deployment step. All the processes of sending approval requests and application deployment in the QA environment are like the previous step. When the application is successfully deployed in the QA environment, the pipeline triggers the production release step. It differs slightly from the previous two steps because it requires manual approval from the authorized person. Once the pipeline receives approval, it first deploys the application to the staging slot. Once the application has been deployed successfully, if the staging slot is running a healthy version of the application, it will be swapped with the production slot.

Azure app service deployment templates were used to create a release. A service connection is created to grant access to the pipeline to publish our applications in the Azure app service. Service connection could be configured to grant limited access to enforce better security. The same service connection is used to configure the tasks. As the first step, a development stage is configured with the development resource group service connection using a Windows machine to publish the applications to the development app service. Since asp.net core is platform independent, we can build the project on a Linux machine and deploy it to a Windows machine or vice versa. Similarly, another two stages are added with the QA and staging service connection using a Windows machine to publish those two applications to QA and staging app service. All these stages are configured to publish the applications simultaneously after successful publishing to the previous stage [81].

## Chapter 4: Implementation

While publishing the application in the production environment, there is a risk that the application will be unavailable until the deployment is completed. To overcome this issue, azure app services deployment slots are used. In this implementation, a staging deployment slot is created in the production app service. With this setup, we can publish the new changes to the staging slot and the application running in the production slot is not affected. The main advantage of the Azure app service is that after the application is published in the staging slot, the application can be swapped to the production slot without any downtime. To configure the staging stage, an additional configuration is required where staging deployment slot is also configured in the app service deployment task. To perform the swap, we need to add one more task called Azure app service management in the staging stage. Azure app service manage task can be used to start, stop, and restart app service. Similarly, the tasks like slot swap, slot delete, installing site extensions, or enabling continuous monitoring can also be performed using Azure app service management [81]. For this implementation, swap slots action is configured to swap the staging slot with the production slot in the existing production app service.

As a safety net, we can add automated approval in each stage so that each deployment can be approved at the development and QA stages after testing and quality assurance. Development approvals could be assigned to the development team members, and QA approvals to the QA manager. While automating the whole process, it will be wise to have manual intervention before publishing the application to production. In each stage where approval is required, azure pipeline sends email notifications to all the reviewers. CD pipeline configured in this stage is illustrated in the following image.

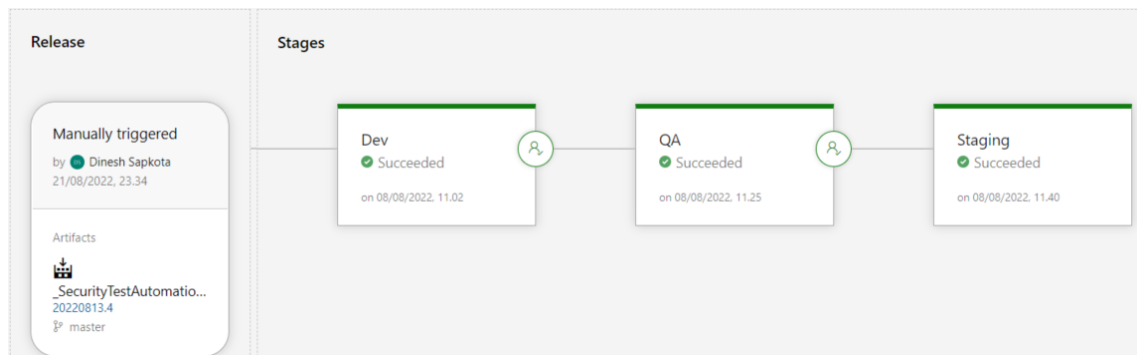


**Figure 4.6: Release pipeline implementation in ADO**

## Chapter 4: Implementation

The above picture illustrates a fully automated CD pipeline flow implemented for this study. The Azure release pipeline provides a comprehensive view of all the stages in the release process, including triggers and approval settings.

Whenever a new build artifact is available in the drop folder release pipeline receives a trigger request to execute the development stage of the pipeline. After the successful execution of the development stage pipeline sends the deployment approval request to the reviewer. When the reviewer approves the request, it triggers the QA deployment stage execution. After its successful execution approval request is sent to the QA manager. Whenever the QA approval is received by the pipeline, it proceeds to the final stage and the execution is paused to get the manual intervention from the project lead to proceed to the production release. At the final stage, when the responsible person resumes the execution, the application will be first published to the staging slot. After that, it will be swapped to the production environment. On the successful execution of the CD pipeline, we could review the process by checking the following flow diagram.



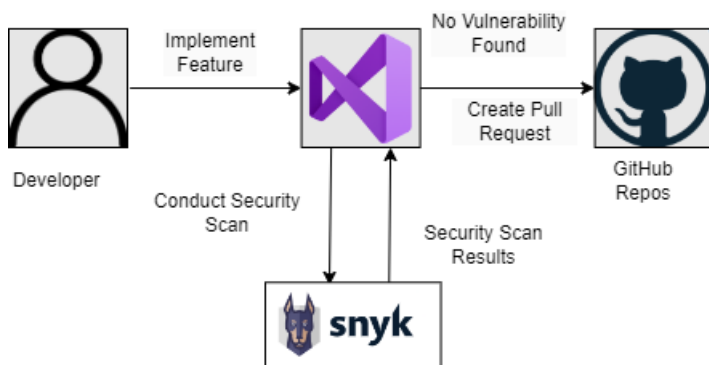
**Figure 4.7: Release pipeline execution flow in ADO**

### 4.3 Shifting Security to the Left

As mentioned previously, the Azure infrastructure test project was utilized to automate the entire SDLC using Azure CI/CD pipelines. In this implementation, the DevOps CI/CD pipeline was used to integrate Snyk security tools into both GitHub and Azure CI pipelines. Recognizing the significance of infrastructure as code (IaC) in ensuring overall application and infrastructure security, IaC security scanning was also incorporated as part of this study.

### 4.3.1 Security Scanning in IDE

Presently, there exists a wide range of security scanning plugins or extensions designed for various IDEs. Integrating security scanning tools directly into IDEs is a significant step towards emphasizing security right from the beginning of the application development process. However, it is worth noting that there is still a scarcity of comprehensive tools available for all programming languages that developers can fully rely on. In this research, a free edition of the Snyk Visual Studio extension was utilized to conduct security scanning. However, it is important to note that the free version of the extension has certain limitations on the number of tests that can be executed. Upon installing the plugin or extension in Visual Studio, the most up-to-date Snyk CLI is also installed concurrently. Both the extension and CLI can be used to carry out security analysis. Both of them support Snyk Open Source and Snyk code scanners. Scans are performed efficiently, yielding high-quality results that include detailed information about vulnerabilities, their severity in CVE format, and recommendations for potential fixes. Additionally, the tool provides references to the corresponding code within the IDE. Automated algorithmic-based fix recommendations are generated for open-source dependency issues, covering both direct and transitive dependencies. Snyk IDE extensions leverage the artificial intelligence and machine learning capabilities of Snyk Code to enhance the analysis and provide comprehensive recommendations. [82] The following diagram illustrates the integration of Snyk into the application development process within Visual Studio, including the security scanning steps performed prior to merging a branch in a Git repository.



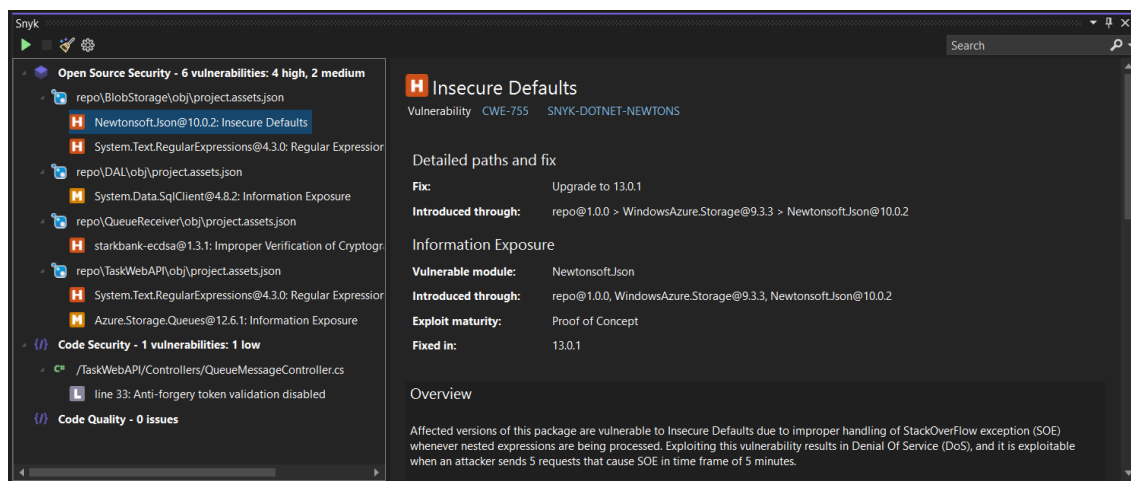
**Figure 4.8: Code Scanning and Pull Request Flow**



## Chapter 4: Implementation

The above figure illustrates software development best practices. As a first step developer implements a feature in Visual Studio. Once the feature is finished, the developer utilizes the Snyk extension to conduct a security scan, examining the application for any potential vulnerabilities. The Snyk extension produces a scan report that can be evaluated by the developer. If the scan report reveals the presence of vulnerabilities, particularly those with critical or high severity scores, appropriate measures must be taken to mitigate them. Only after fixing those security issues developer should push the changes to git repositories.

As mentioned previously, a test application was created for this research using NuGet packages that were available in 2019. This choice was made to gain insight into newly discovered vulnerabilities that emerge quickly. Despite the limited features and dependencies of this application, when conducting the security scan using the Snyk extension, a total of four high-risk, two medium-risk, and one low-risk vulnerabilities were detected. This indicates that within five years, four high-risk and two medium-risk vulnerabilities have been identified. In the meantime, considering a significantly bigger application, the number of vulnerabilities will also increase significantly. The following snapshot illustrates the results generated by the Visual Studio Snyk extension.



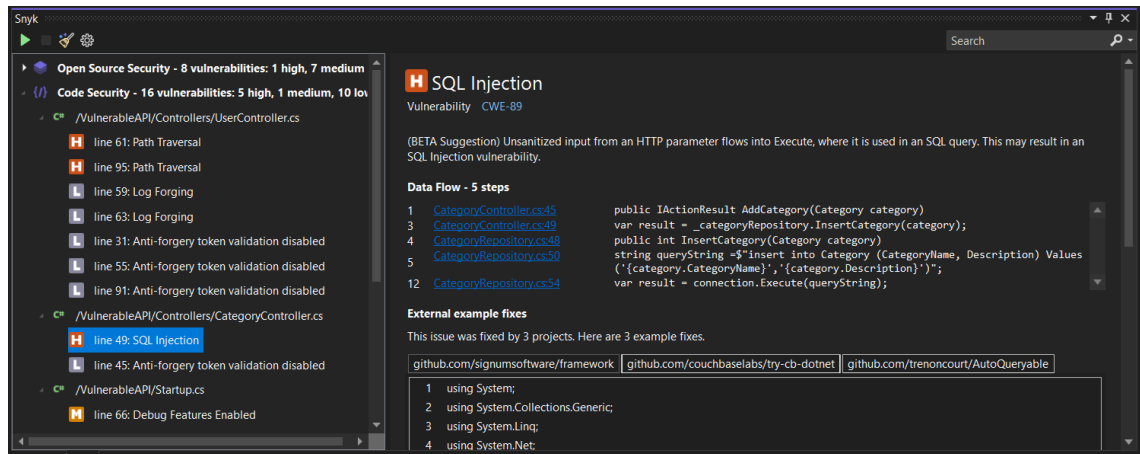
**Figure 4.9: Open-Source vulnerabilities discovered in the test project.**

In the above figure vulnerabilities are divided into Open-Source Security, Code Security and Code Quality categories. In this section, an example Open-Source Security issue is selected to check the generated result. It generates a quality report pinpointing the location of the existing vulnerabilities. The information provided in the result's detailed

## Chapter 4: Implementation

section includes the severity level, CWE id, the vulnerable dependency, an overview of the vulnerability, and recommended fixes. This specific vulnerability was chosen as an illustrative example because it highlights an important case. The main NuGet package used in the application, "Windows.Azure.Storage," had an underlying dependency on the "Newtonsoft.json" NuGet package. The version of "Newtonsoft.json" being utilized by "Windows.Azure.Storage" was 10.0.2, which contained vulnerabilities related to Insecure Defaults. These vulnerabilities had the potential to be exploited for a Denial-of-Service (DoS) attack. The scan result indicates that the identified vulnerability has been resolved in version 13.0.1 of the "Newtonsoft.json" package. Updating the affected NuGet package to this version is recommended to mitigate the vulnerability. These types of vulnerabilities can be challenging for developers to detect, and security scanning extensions like Snyk prove valuable in ensuring security measures are implemented early in the SDLC.

Similarly, in the following screenshot, an example code security issue is picked to check the quality of the generated result.



**Figure 4.10: Code Security Vulnerabilities discovered in the test project.**

To conduct this portion of the study, a security vulnerability found in the vulnerability test project was utilized as an illustrative example of code vulnerability. As in the Open-Source Security scan result, it also pinpoints the vulnerability in the source code along with the complete flow of the input data in 5 steps in the source code. The detailed information section contains vulnerability severity, vulnerability name, CWE id and an overview of the vulnerability. Upon examining the data flow steps, it becomes evident

## Chapter 4: Implementation

---

that the user input is directly incorporated into the SQL query, rendering the application susceptible to SQL Injection attacks. The scan result also suggests some example fixes, but in this current context, they were not accurate fixes.

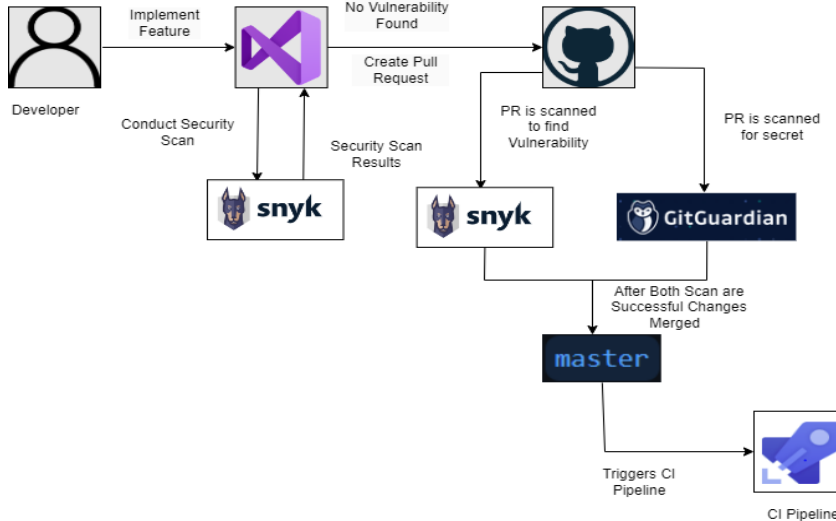
### 4.3.2 Security Scanning in SCM System

Snyk integrations are compatible with the majority of available Source Code Management (SCM) systems. In this study, Snyk was integrated into the GitHub repository [83]. The integration of Snyk with GitHub allows for continuous security scanning of the repository. It aids in the identification of vulnerabilities in open-source dependencies and provides recommendations for fixing them, including upgrading the dependencies to the mitigated versions. The Snyk UI provides the option to integrate a GitHub repository for Snyk's services. It also allows for additional customization, such as project-level reports, project monitoring, automatic pull requests for fixes, commit signing, and pull request security testing. Enabling pull request security scanning in GitHub is an essential component of achieving comprehensive and automated DevSecOps integration.

Whenever a pull request is initiated in the repository, Snyk performs a vulnerability test on the new changes and reports the test status back to GitHub. It is considered a best practice to consider applications with vulnerabilities of critical and high severity scores as insecure. Hence, in this study, the security checks are configured to mark the pull request as failed if the application contains vulnerabilities with severity scores classified as critical or high [84]. The Snyk PR checks feature allows us to examine each pull request made for merging changes into the master branch. This feature assists developers in addressing security issues in case they accidentally push code to the repository without conducting an IDE scan. By utilizing pull request checks, vulnerable code is prevented from being merged into the master branch, thereby avoiding the automatic triggering of the CI pipeline. It is important to note that the current Snyk SCM integration does not support static application security testing of source code [85]. As the existing version of Snyk Code was unable to identify sensitive information stored in the configuration, GitGuardian was integrated into GitHub to address this limitation. GitGuardian conducts secret checks during pull requests and notifies the

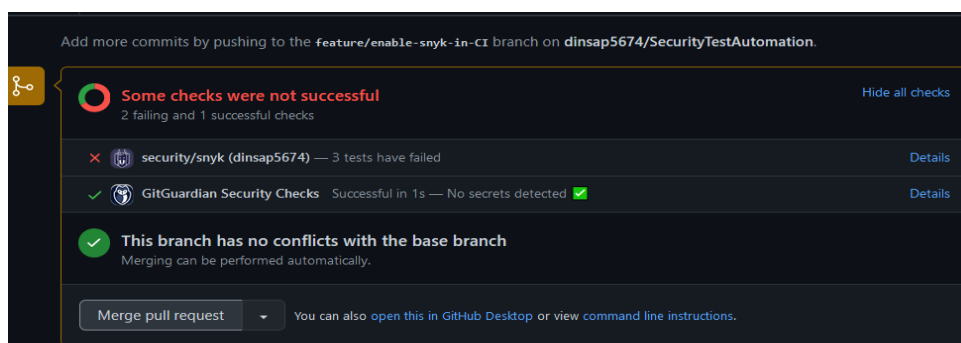
## Chapter 4: Implementation

designated contact person if any secrets are detected. It also fails the pull request in such cases. The implemented flow in this study is shown in the following image.



**Figure 4.11: Security scan and execution flow in GitHub**

Upon creating a new pull request in the GitHub repository, the Snyk security scan and GitGuardian secret scan are automatically triggered. The changes can only be merged into the master branch once both scans report a successful status. After the changes are merged, the CI pipeline is triggered to build the artifacts. The following figure illustrates the triggered Snyk and GitGuardian scans along with the reported scan statuses.

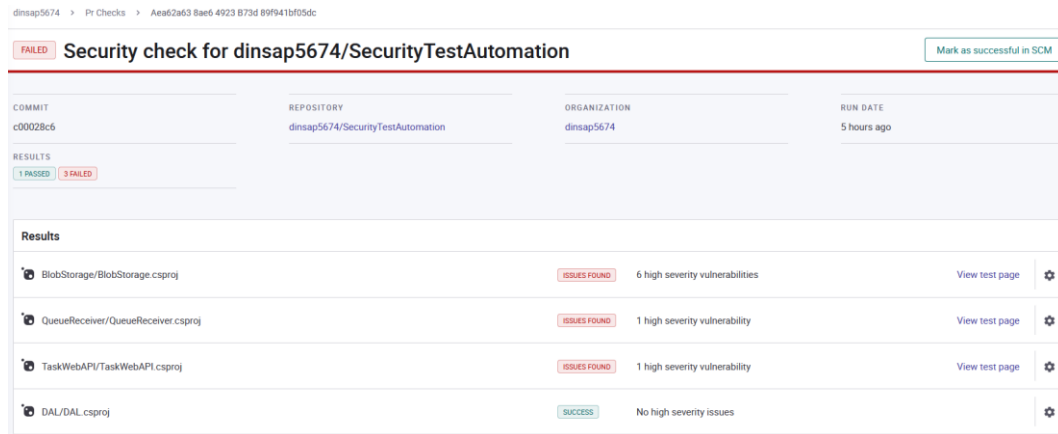


**Figure 4.12: GitHub pulls request execution flow.**

In the test production application, all the secrets were securely stored in the Azure Key Vault and accessed using managed identity. As a result, the GitGuardian security check did not detect any secrets and reported a successful status. However, since the application was developed using NuGet packages from 2019, which are known to have

## Chapter 4: Implementation

vulnerabilities, as discussed earlier, the Snyk security scan failed and caused the pull request to fail. Clicking on "details" provides additional information about the reported security issues. The provided image displays a list of all the discovered vulnerabilities.



**Figure 4.13: Pull Requests dependency security scan details.**

All the detailed information about the vulnerabilities can be found on the Snyk account PR check page. The provided image displays a list of project dependencies along with the vulnerabilities detected in those dependencies. Upon closer inspection, it can be observed that there are variations in the number of vulnerabilities reported. This is because the GitHub repository is configured to only check for high and critical severity vulnerabilities. In this particular case, the BlobStorage library identified six high severity vulnerabilities in the PR check result. However, upon further investigation of the vulnerabilities, it was found that these vulnerabilities were associated with Windows.Azure.Storage NuGet package. The Snyk IDE security result, on the other hand, only displayed the two most severe vulnerabilities discovered in the same NuGet package. The following image showcases a privilege escalation vulnerability that was discovered in the same package.

## Chapter 4: Implementation

The screenshot displays a GitHub PR check for a vulnerability in the System.Net.Http package. The title is "System.Net.Http - Privilege Escalation". Below the title, there are several identifiers: VULNERABILITY, CWE-269, CVE-2017-0249, CVSS 7.3, and SNYK-DOTNET-SYSTEMNETHTTP-60047. A "HIGH" severity badge is visible. A message states: "Runtime Vulnerability: This vulnerability is probably caused by the installed .NET Framework/runtime version. View Docs". The "Introduced through" field shows "WindowsAzure.Storage@9.3.3". The "Fixed in" field shows "System.Net.Http@4.1.2, @4.3.2". The "Exploit maturity" is "NO KNOWN EXPLOIT". A "Detailed paths" section shows the path: "project@undefined > WindowsAzure.Storage@9.3.3 > NETStandard.Library@1.6.1 > System.Net.Http@4.3.0". The "Security information" section lists factors contributing to the scoring: "Snyk: CVSS 7.3 - High Severity" and "NVD: Not available. NVD has not yet published its analysis." The "Overview" section provides a brief description of System.Net.Http and notes that affected versions are vulnerable to Privilege Escalation due to failing to properly sanitize web requests.

**Figure 4.14: Privilege escalation vulnerability discovered in GitHub PR checks.**

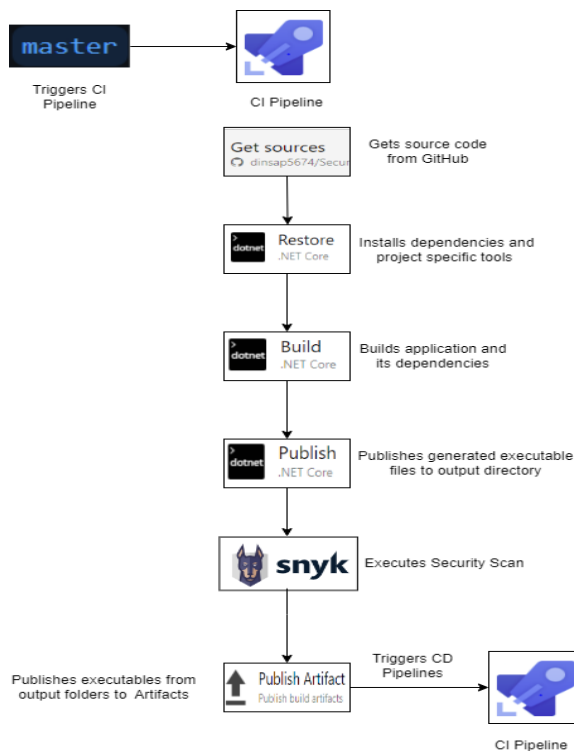
Upon reviewing the specifics, it becomes evident that this vulnerability emerged in version 9.3.3 of Windows.Azure.Storage. This package incorporates an internal dependency on System.Net.Http version 4.1.2, which contains vulnerabilities such as Denial-of-Service (DoS), improper certification validation, privilege escalation, and information exposure. The provided image highlights the specific details of the privilege escalation vulnerability. To address these vulnerabilities, it is recommended to remove the deprecated Windows.Azure.Storage NuGet package and instead install the necessary Azure.Storage.\* NuGet packages.

### 4.3.3 Security Scanning in Build Pipeline

Snyk offers integration with various CI/CD platforms, including Azure Pipelines, which was examined in this study. Azure Pipelines can integrate Snyk security as one of its tasks. The security scan using Snyk can be executed after the creation and storage of executable files in the archive folder. Within the Azure interface, a dedicated Snyk security task is available, which can be inserted into the pipeline and customized as per requirements. In the CI pipeline, the Snyk test can be integrated alongside other routine tasks before publishing the executables to the artifacts store. Once the build task successfully generates the executable files, the Snyk security test is triggered to identify any security vulnerabilities in the application. The scan results can be accessed both in

## Chapter 4: Implementation

the Azure Pipelines output and the Snyk interface. For this particular study, the Snyk security task was configured to fail if it detected vulnerabilities with critical and high severity. Consequently, when the CI pipeline fails, no executables are published in the artifacts store. The CD pipeline is only triggered when new artifacts are successfully published. As a result, the vulnerable application is stopped in the CI pipeline, ensuring that the production application remains safeguarded against vulnerabilities. The following figure illustrates the execution flow from the master branch to the CD pipeline.



**Figure 4.15: Snyk security test in CI pipeline**

To enable the merging of vulnerable application code into the master branch, the GitHub Snyk PR check is deactivated. Once these changes are merged, the CI pipeline is triggered. The subsequent steps, including the creation of separate executable files for the Web API and Web Job projects and their publication in the archive output folder, remain consistent with the previously discussed CI pipeline. Following that, the Snyk security test utilizes the \*.sln file to conduct a security scan and identify any existing vulnerabilities. If the Snyk test identifies vulnerabilities with a severity score of critical or high, the CI pipeline fails to push any artifacts to the drop folder. As the CD pipeline

## Chapter 4: Implementation

is activated only when there are changes in the artifacts, these modifications will not be deployed in the production environment. The following code snippet exemplifies the Snyk security test configuration employed in this study.

```
#For windows
- task: SnykSecurityScan@1
  inputs:
    serviceConnectionEndpoint: 'Snyk Service Connection'
    testType: 'app'
    targetFile: 'D:\a\1\s\SecurityAutomation.sln'
    monitorWhen: 'always'
    failOnIssues: true
  enabled: true
```

### Code Snippet 4.17: Snyk security test task configuration

The given code snippet introduces the task named SnykSecurityScan, which utilizes the Snyk Service Connection to perform the scan. It is set up to test applications using the SecurityAutomation.sln file. The configuration ensures monitoring is consistently enabled, and the pipeline fails when issues are detected, with security scanning enabled. Upon executing this pipeline in the present context, the following outcome will be observed in Azure.

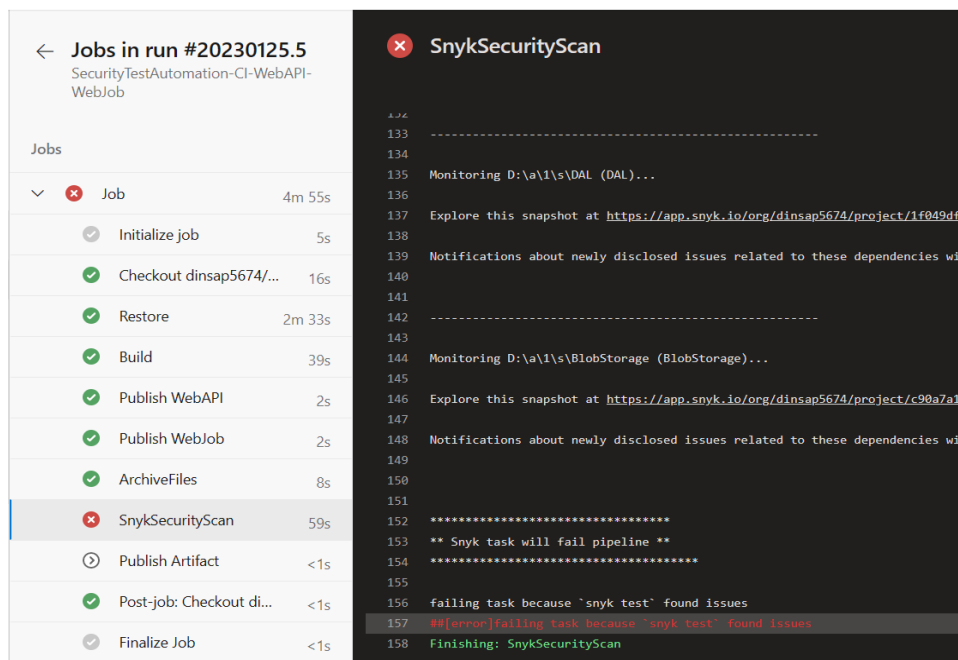


Figure 4.16: CI pipeline execution flow in Azure



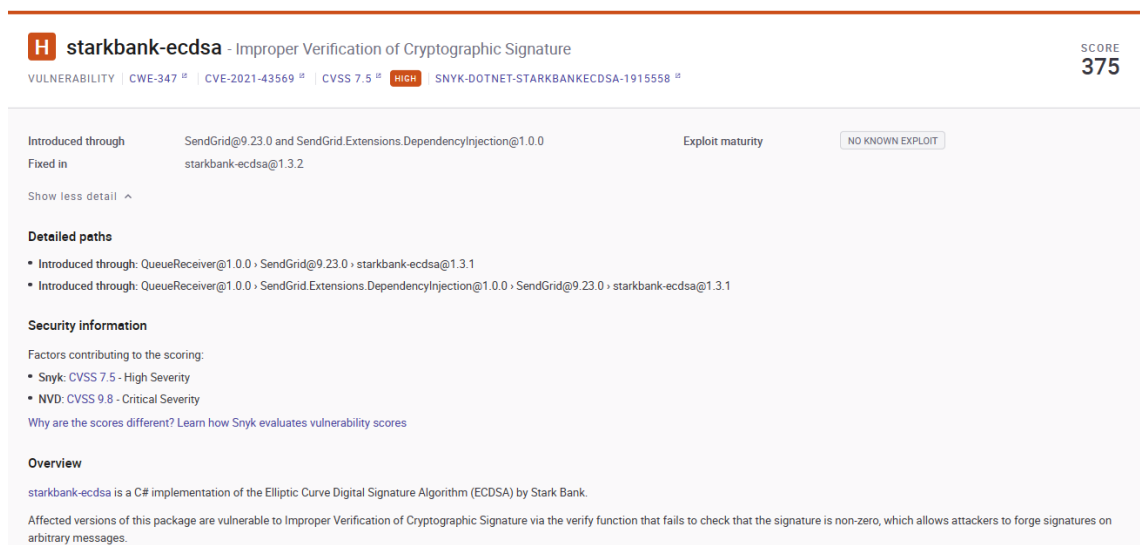
## Chapter 4: Implementation

The execution flow of the Azure CI pipeline reveals that the pipeline encountered a failure during the execution of the Snyk Security scan, causing the subsequent task to be skipped. The execution log provides a concise overview of the issue, and further details can be accessed by clicking the provided snapshot link. Additionally, the Snyk report within the pipeline allows for a comprehensive examination of all identified vulnerabilities. The accompanying image visually demonstrates the condensed result within Azure.



**Figure 4.17: Brief information about security issues in azure build pipeline**

The image summarises the scanned projects and the total number of vulnerabilities detected. By scrolling further in the report, more information about the identified vulnerabilities can be found. The subsequent image showcases one of the vulnerabilities that were discovered.



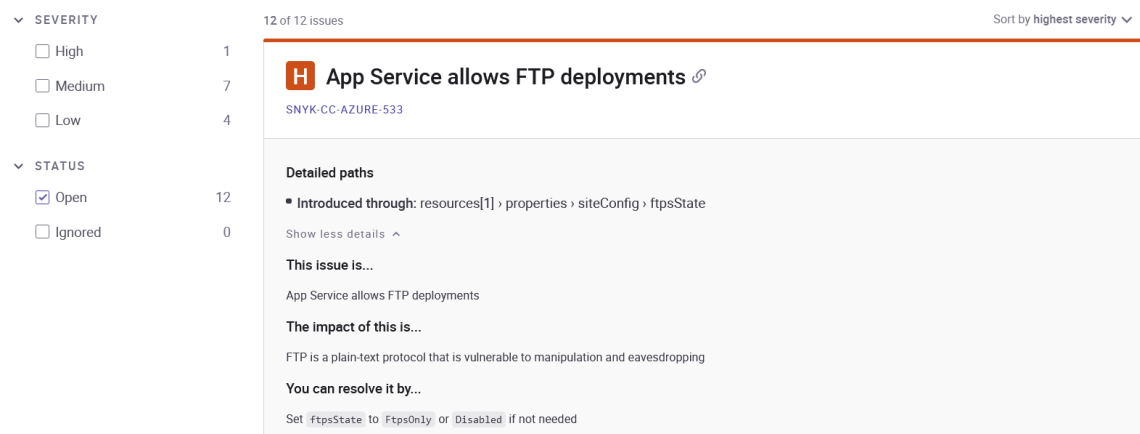
**Figure 4.18: Vulnerability Detected - Improper Verification of Cryptographic Signature in Existing Dependency.**

## Chapter 4: Implementation

The vulnerability in the above picture was detected in the SendGrid v9.23.0 NuGet package, specifically in its inner dependency, starkbank-ecdsa v1.3.1. This vulnerability is related to the usage of the Elliptic Curve Digital Signature Algorithm within the package. To address this issue, it is recommended to update the package to version 1.3.2 or a higher version. It is important to highlight that across all versions of Snyk, consistent and actionable results have been produced.

### 4.3.4 Vulnerability Scanning in IaC

Snyk's Infrastructure as Code (IaC) functionality can be employed to enhance the security of cloud infrastructure, both before and after deploying cloud resources. Snyk offers a comprehensive collection of security rules designed to identify misconfigurations in IaC. By utilizing Snyk's IaC security scan, we can identify security vulnerabilities within the current IaC [86]. In this research, the ARM template was utilized to create the necessary Azure infrastructure. Similarly, the focus was solely on scanning the IaC template file for security before deployment. As previously mentioned, the ARM template contained the deployment of Azure Key Vault, SQL Server, Storage Account, App Service, and App Service plan resources. The resources were created using the default template, and customization was performed to fulfil specific requirements. During the execution of the Snyk IaC security scan, the following vulnerabilities were identified.



The screenshot displays the Snyk IaC security scan results interface. On the left, there are filters for SEVERITY and STATUS. The SEVERITY filter shows 1 High, 7 Medium, and 4 Low issues. The STATUS filter shows 12 Open and 0 Ignored issues. The main content area shows a list of 12 issues, with the top issue being a High severity issue: 'App Service allows FTP deployments' (SNYK-CC-AZURE-533). The detailed view for this issue shows the path: 'resources[1] > properties > siteConfig > ftpsState'. The issue description states: 'App Service allows FTP deployments'. The impact is: 'FTP is a plain-text protocol that is vulnerable to manipulation and eavesdropping'. The resolution is: 'Set ftpsState to FtpsOnly or Disabled if not needed'.

**Figure 4.19: Snyk IaC security scan results.**

The provided image demonstrates the detection of 12 vulnerabilities by Snyk. Among these vulnerabilities, one had a high severity level, seven had medium severity, and four

## Chapter 4: Implementation

---

were classified as low severity. Regarding IaC security scanning, the generated result included sufficient information about the identified issues and provided mitigation recommendations, which would be valuable for developers. The vulnerability labelled as "App service allows FTP deployments" was classified as high severity. This vulnerability could expose the application to manipulation and eavesdropping attacks due to the inherent lack of security in the FTP (File Transfer Protocol) protocol, which transmits data in plain text. To address this issue, the recommended fix is to set the value of the "ftpsState" parameter to "FtpsOnly". However, in the provided ARM template, this setting was not defined resulting in the use of the default configuration which was FTP. In addition, there were seven vulnerabilities classified as medium severity. These vulnerabilities include the following issues: SAS token can be used over insecure HTTP, Function App does not enforce HTTPS, Azure App Service allows HTTP traffic, Storage Account does not enforce latest TLS, Storage Account geo-replication disabled, use two or more App Service Plan instances and App Service remote debugging enabled. It is recommended to address most of the medium-severity risks before deployment due to their relatively easier fix. Neglecting to do so would leave the Azure services vulnerable to a wide range of potential attacks. The remaining four low-severity vulnerabilities include Storage Account Blob service delete disabled, App Service HTTP/2 disabled, App Service mutual TLS disabled, and SQL Server auditing disabled. Among these, three vulnerabilities do not pose any significant security risks, while the remaining one has minimal security impact. It is advisable to fix even the vulnerability with minimal risk, considering the simplicity of the fix.

# Chapter 5

## 5 Results and Evaluation

The IT security industry has conventionally followed a reactive approach to mitigate security vulnerabilities after security breaches are discovered. Vulnerabilities are categorized based on their risk severity, such as critical, high, medium, and low, using CVE identifiers. Security guidelines recommend mitigating critical and high-risk vulnerabilities based on the specific requirements of the application. However, this study demonstrates that most of the existing applications are unable to mitigate even all the critical vulnerabilities.

To evaluate the situation, a test production application was developed using older frameworks and libraries from 2019, following recommended security best practices. The application was then tested for security vulnerabilities using the Snyk Visual Studio extension. The test results indicated that the application did not contain any custom code vulnerabilities. However, the tool detected four high-risk and three medium-risk vulnerabilities in the NuGet packages used by the application. Despite the limited features of the test application, it relied on 17 different NuGet packages. The results revealed that approximately 41% of the used packages had new vulnerabilities discovered within five years. Interestingly, most of the vulnerabilities identified in this study were found in NuGet packages developed by Microsoft. Generally, the Microsoft ecosystem is considered to have fewer open-source dependencies. Detecting vulnerabilities in dependencies can be challenging due to the complex hierarchical structure of inner dependencies. Table 5.1 lists the discovered vulnerabilities, along with the actual dependencies used, their inner dependencies, and their severity scores.

## Chapter 5: Results and Evaluation

Actual Dependencies	Inner Dependencies	Severity
Windows.Azure.Storage v9.3.3	Newtonsoft.Json v10.0.2	High
	System.Text.RegularExpressions v4.3.0	High
SendGrid v9.23.0	Starkbank-ecdsa v1.3.1	High
Azure.Identity v1.2.2	System.Test.RegularExpressions v4.3.0	High
System.Data.SqlClient v4.8.2		Medium
Swashbuckle.AspNetCore v6.1.3	Swashbuckle.AspNetCore.SwaggerUI v6.1.3	Medium
Azure.Storage.Queues v12.6.1		Medium

**Table 5.1: Vulnerable packages discovered in the test production application.**

The study reveals that modern applications heavily rely on dependencies, with approximately 78% of the code coming from these external dependencies or libraries. Only a small portion of the application-specific custom code is written by the developer. Due to the complexity and volume of dependencies, manually checking for vulnerabilities in each of the dependencies is challenging. However, existing vulnerability scanning tools for open-source software (OSS) have proven to be effective in detecting such vulnerabilities. To evaluate the performance of Snyk code and Snyk open-source tools in detecting dependencies, ten known dependencies with vulnerabilities were tested, and both tools successfully detected all of them.

In general, the study emphasizes the significance of automated vulnerability detection tools. Still, it is recommended to use an OSS vulnerability scanner because those OSS vulnerabilities are difficult to detect manually. The integration of these tools into the development workflow helps us to ensure the security of applications throughout the development and deployment process. To further streamline the security integration in DevOps CI/CD pipelines, the Snyk open-source tool was integrated into both GitHub and Azure build pipelines. However, it is important to note that Snyk open-source tool does not support secrets detection. Due to that reason, GitGuardian secret scanner was also integrated into GitHub to prevent the merging of source code containing secrets

## Chapter 5: Results and Evaluation

---

into the master branch. Similarly, Snyk code tool successfully identified hard-coded secrets but did not detect secrets embedded within configuration files. To evaluate the performance of both tools, a vulnerable application with five hard-coded secrets along with five secrets in configuration files was tested. Table 5.2 shows the secret detection results using these two tools.

Secret Types	Total Known Secrets	GitGuardian	Snyk Code
Hardcoded secrets	5	5	5
Secrets in configuration files	5	5	0

**Table 5.2: Secrets detection results of different scanning tools**

As we can see in Table 5.2, the test result clearly showed that GitGuardian has a 100% detection rate, whereas Snyk code only detected hardcoded secrets. Similarly, GitGuardian integration supports automatic rejection of a developer's pull request if any secrets are detected in the source code. Because of this, it is recommended to integrate the GitGuardian secret scanner into the SCM system to detect any exposed secrets.

Although developing a secure application is important, it is equally crucial to ensure the security of the underlying infrastructure on which the application is hosted. This is because vulnerabilities in the infrastructure can make even a secure application vulnerable. Therefore, it is necessary to establish a secure infrastructure as well. As previously discussed, an IaC implementation was created for deploying the test production application. During the creation of Azure resources using IaC, certain known vulnerable protocols, such as FTP, HTTP, and TLS, were intentionally misconfigured to evaluate the performance of the Snyk CLI tool in detecting security vulnerabilities. Additionally, the scanning tool also discovered other vulnerabilities that were not initially identified during the creation and configuration of Azure resources. The tool identified one high-risk, seven medium-risk, and four low-risk vulnerabilities. Table 5.3 lists the high and medium risks identified by the tool.

## Chapter 5: Results and Evaluation

Azure Resource	Discovered Vulnerability	Severity
App Service	App Service allows FTP deployments	High
Storage Account	SAS token can be used over insecure HTTP	Medium
Function App	Function App does not enforce HTTPS	Medium
App Service	App Service allows HTTP traffic	Medium
Storage Account	Storage Account does not enforce latest TLS	Medium
Storage Account	Storage Account geo-replication disabled	Medium
App Service Plan	Use two or more App Service Plan instances	Medium
App Service	App Service remote debugging enabled	Medium

**Table 5.3: Vulnerabilities detected in IaC implementations.**

The scanning tool successfully detected the specific protocol-related threats that were intentionally configured to test its performance. The infrastructure was deployed specifically for the purpose of the study, and certain decisions, such as disabling geo-replication and using a single app service plan, were made to minimize costs. However, the scanning tool was able to identify the risk of potential data loss in the event of a disaster at the hosted location, as well as the potential unavailability of the app service when only one app service plan is used. Furthermore, the tool detected the default configuration of the app service, which enabled remote debugging, and warned developers that it might expose the application to unnecessary risk.

SAST tools currently available are not considered entirely reliable due to their known weaknesses, such as high false positive rates and low detection rates. Moreover, these tools cannot be integrated into SCM systems and CI/CD pipelines. Therefore, in this study, the main focus was on identifying a set of tools that can be utilized within the IDE to empower developers in enforcing security.

Among the available options, Snyk Code was chosen as an extension for Visual Studio and used for evaluating its performance as a SAST tool. To assess its effectiveness, a vulnerable test project was created, which contained eight security vulnerabilities

## Chapter 5: Results and Evaluation

---

according to OWASP Web API Security guidelines. Snyk Code successfully detected only four out of the eight vulnerabilities, resulting in a detection rate of approximately 50%. While these results indicate that complete reliance on the tool may not be feasible, it can still serve as a valuable resource for developers to improve security posture. Table 5.4 presents the different vulnerabilities and their detection using Snyk Code.

Known Vulnerabilities	Snyk Code Detection
Broken Object Level Authorization	No
Broken User Authentication	Yes
Excessive Data Exposure	Yes
Lack of Resource and Rate Limiting	No
Broken Function Level Authorization	No
Mass Assignment	No
Security Misconfiguration	Yes
Injection Vulnerability	Yes

**Table 5.4: Snyk code vulnerability detection results of OWASP Web API 10 list**

The outcome provides a clear demonstration of the potential of SAST tools in empowering developers to enforce security measures. However, it is important to note that existing tools cannot be fully depended upon to guarantee comprehensive security. Nevertheless, incorporating these tools remains advantageous as they are capable of detecting a substantial number of vulnerabilities, contributing to improved overall security practices.

Overall, the Snyk tools produce comprehensive vulnerability reports that include accurate information and recommended mitigation measures. In all test cases, following the provided recommendations successfully mitigated the identified security risks. The generated reports by Snyk tools consistently maintain a high level of quality.



# Chapter 6

## 6 Conclusion

This study involved an extensive exploration of multiple research questions related to the integration and automation of security checks within the DevOps framework. The main focus was directed towards investigating the implementation and assessing the effectiveness of security scanning tools. The primary goal of this research was to examine the feasibility of incorporating automated security checks into DevOps practices.

Regarding the first research question, “What is the availability of supported security automation tools suitable for DevOps?”, the study clearly demonstrates that there are a wide range of security automation tools available that can be integrated into fully automated DevOps CI/CD pipelines. However, these tools have not yet received a definitive recommendation from the security community as efficient security scanning tools. Nevertheless, they offer the opportunity to shift security practices to earlier stages of the SDLC. In this particular study, Snyk was utilized among the available tools.

For the second research question, “How effectively do these tools cover various security aspects?”, based on the evaluation results, it appears that Snyk open source can be highly dependable for detecting vulnerabilities in open-source dependencies, i.e., NuGet packages. It can be seamlessly integrated into SCM systems and CI/CD pipelines. Additionally, GitGuardian can be integrated into SCM systems to identify secrets within source code. Snyk Code can be used as an IDE extension to detect vulnerabilities in custom code. These three tools address the most critical security considerations from software development perspectives. By sticking to software development best practices and leveraging these tools, developers could be empowered to create secure software. Continuous research and development efforts are ongoing to enhance the performance

## Chapter 6: Conclusion

---

of SAST tools, which are gradually improving over time. Furthermore, Snyk CLI can be employed to detect security misconfigurations in IaC, guaranteeing the deployment of secure infrastructure.

Furthermore, the third research question, “What is the accuracy and reliability of the security scan results they provide?” was also investigated in this study. Based on the performance outcomes of the scanning tools, the integration of Snyk open source and GitGuardian into a completely automated CI/CD pipeline guarantees the utilization of secure dependencies and the absence of confidential information in GitHub repositories. Snyk Code, while not completely perfect with a 50% detection rate, can also be utilized to identify vulnerabilities in custom code. The study findings also indicate that Snyk CLI can be relied upon to ensure infrastructure security.

The fourth research question, “Can security automation be implemented early in the SDLC?” was also explored in this study. According to the findings, these suggested tools have the potential to be employed right from the start of the SDLC. By adopting the proposed guidelines and utilizing the recommended tools, developers can seamlessly integrate security automation throughout the entire development process, from the IDE to the CD pipeline. This shift to the left enables developers to perform security scanning after completing each new feature, allowing them to identify any vulnerabilities in the source code promptly.

Finally, for the last research question, “Will it contribute to faster software delivery and cost reduction?”. Based on the findings of the study, upon reviewing the scan results, developers can gain a deeper understanding of the vulnerabilities and follow the recommended actions to mitigate them. This proactive approach is both efficient and cost-effective, as the vulnerabilities are addressed while the implementation details are still fresh in the developers' minds. Furthermore, this approach relieves some of the workload of security experts, who can then focus on addressing security aspects not covered by automated security integrations.

To prevent accidental and careless merging of vulnerable source code into the GitHub repository, the integration of Snyk open source and GitGuardian provides an additional layer of security. Similarly, integrating Snyk open source into the CI/CD pipeline

## Chapter 6: Conclusion

---

ensures that vulnerable applications are not deployed from the pipelines. This comprehensive approach empowers software development teams to prioritize and ensure software security throughout the development lifecycle. Organizations can leverage this approach to streamline the security audit process, eliminating potential bottlenecks in DevOps practices. By implementing this approach, small and medium-sized organizations can enhance their overall security posture.

### 6.1 Future Works

The proposed framework currently does not include more efficient DAST tools. However, integrating DAST tools into the framework can potentially address the limitations of the previously discussed SAST tools. Therefore, it is recommended to include DAST tool integration for further improvement.

This study primarily focused on the backend development aspects of web applications using the C# programming language and Azure cloud services. Moving forward, the framework can be expanded to include research on different security aspects related to front-end application development. While this study specifically examined the Microsoft ecosystem, the framework can be further extended to accommodate other software development ecosystems as well.

# References

- [1] “Internet Growth Statistics 1995 to 2022 - the Global Village Online,” *Internet World Stats*, 2022. <https://www.internetworldstats.com/emarketing.htm> (accessed Jun. 16, 2022).
- [2] “World Population Clock: 8.0 Billion People (2022) - Worldometer,” *WorldOMeter*, 2022. <https://www.worldometers.info/world-population/> (accessed Jun. 16, 2022).
- [3] “Key Internet Statistics to Know in 2022 (Including Mobile) - BroadbandSearch,” *Broadband Search*, 2022. <https://www.broadbandsearch.net/blog/internet-statistics> (accessed Jun. 16, 2022).
- [4] Steve Morgan, “2019/2020 Cybersecurity Almanac: 100 Facts, Figures, Predictions and Statistics,” 2019. Accessed: Jun. 17, 2022. [Online]. Available: <https://cybersecurityventures.com/cybersecurity-almanac-2019/>
- [5] Morgan Steve, “Cybercrime To Cost The World \$10.5 Trillion Annually By 2025,” *Cybercrime Magazine*, Nov. 13, 2020. <https://cybersecurityventures.com/cybercrime-damage-costs-10-trillion-by-2025/> (accessed Jun. 16, 2022).
- [6] “Software vulnerability snapshot - an analysis by Synopsys Application Security Testing Services,” 2021.
- [7] “Open Source Security and Risk Analysis Report,” 2021.
- [8] “What is DevOps?,” *IBM*. <https://www.ibm.com/cloud/learn/devops-a-complete-guide> (accessed Jun. 21, 2022).
- [9] A. Ravichandran, K. Taylor, and P. Waterhouse, *DevOps for Digital Leaders*. 2016. doi: 10.1007/978-1-4842-1842-6.
- [10] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” *IEEE Softw*, vol. 33, no. 3, pp. 94–100, May 2016, doi: 10.1109/MS.2016.68.
- [11] G. Siewruk, W. Mazurczyk, and A. Karpiński, “Security assurance in Devops methodologies and related environments,” *International Journal of Electronics and Telecommunications*, vol. 65, no. 2, pp. 211–216, 2019, doi: 10.24425/ijet.2019.126303.
- [12] “What is DevSecOps?,” *IBM*. <https://www.ibm.com/cloud/learn/devsecops> (accessed Jun. 21, 2022).
- [13] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect’s Perspective*. in SEI Series in Software Engineering. Pearson Education, 2015.
- [14] M. Steve, “The Origins of DevOps: What’s in a Name?,” *DevOps.com*, 2018. <https://devops.com/the-origins-of-devops-whats-in-a-name/> (accessed Jul. 04, 2022).
- [15] M. Meyer, “Continuous Integration and Its Tools,” *IEEE Softw*, vol. 31, no. 3, pp. 14–16, May 2014, doi: 10.1109/MS.2014.58.
- [16] G. G. Claps, R. Berntsson Svensson, and A. Aurum, “On the journey to continuous deployment: Technical and social challenges along the way,” in *Information and Software Technology*, Elsevier B.V., 2015, pp. 21–31. doi: 10.1016/j.infsof.2014.07.009.

## References

---

- [17] M. Leppänen *et al.*, “The highways and country roads to continuous deployment,” *IEEE Softw*, vol. 32, no. 2, pp. 64–72, Mar. 2015, doi: 10.1109/MS.2015.50.
- [18] M. Virmani, “Understanding DevOps & bridging the gap from continuous integration to continuous delivery,” in *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, IEEE, May 2015, pp. 78–82. doi: 10.1109/INTECH.2015.7173368.
- [19] K. Morris, *Infrastructure as Code MANAGING SERVERS IN THE CLOUD*, First. 2016.
- [20] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D. A. Tamburri, “DevOps: Introducing infrastructure-as-code,” in *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017*, Institute of Electrical and Electronics Engineers Inc., Jun. 2017, pp. 497–498. doi: 10.1109/ICSE-C.2017.162.
- [21] Stephen J. Bigelow, “What is Infrastructure as Code?,” *TechTarget*. <https://www.techtarget.com/searchitoperations/definition/Infrastructure-as-Code-IAC> (accessed Dec. 03, 2022).
- [22] B. Chen, “Improving the Software Logging Practices in DevOps,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, IEEE, May 2019, pp. 194–197. doi: 10.1109/ICSE-Companion.2019.00080.
- [23] T. Dyba and T. Dingsoyr, “What Do We Know about Agile Software Development?,” *IEEE Softw*, vol. 26, no. 5, pp. 6–9, Sep. 2009, doi: 10.1109/MS.2009.145.
- [24] M. Gokarna and R. Singh, “DevOps: A Historical Review and Future Works,” in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, IEEE, Feb. 2021, pp. 366–371. doi: 10.1109/ICCCIS51004.2021.9397235.
- [25] Van Wyk, Kenneth R., and G. Mcgraw, “Bridging the gap between software development and information security,” *IEEE Secur Priv*, vol. 3, no. 5, pp. 75–79, 2005, doi: 10.1109/MSP.2005.118.
- [26] Tony Hsu, *Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps*, First. Packet Publishing Ltd, 2018.
- [27] V. Mohan and L. Ben Othmane, “SecDevOps: Is It a Marketing Buzzword? - Mapping Research on Security in DevOps,” in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, IEEE, Aug. 2016, pp. 542–547. doi: 10.1109/ARES.2016.92.
- [28] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, “Challenges and solutions when adopting DevSecOps: A systematic review,” *Inf Softw Technol*, vol. 141, p. 106700, Jan. 2022, doi: 10.1016/J.INFSOF.2021.106700.
- [29] M. A. Akbar, K. Smolander, S. Mahmood, and A. Alsanad, “Toward successful DevSecOps in software development organizations: A decision-making framework,” *Inf Softw Technol*, vol. 147, p. 106894, Jul. 2022, doi: 10.1016/J.INFSOF.2022.106894.
- [30] R. N. Rajapakse, M. Zahedi, and M. A. Babar, “An Empirical Analysis of Practitioners’ Perspectives on Security Tool Integration into DevOps,” in *Proceedings of the 15th ACM / IEEE International Symposium on Empirical*

## References

---

- Software Engineering and Measurement (ESEM)*, New York, NY, USA: ACM, Oct. 2021, pp. 1–12. doi: 10.1145/3475716.3475776.
- [31] A. Ibrahim, A. H. Yousef, and W. Medhat, “DevSecOps: A Security Model for Infrastructure as Code over the Cloud,” in *MIUCC 2022 - 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 284–288. doi: 10.1109/MIUCC55081.2022.9781709.
- [32] W. Baker and L. Wallace, “Is Information Security Under Control?: Investigating Quality in Information Security Management,” *IEEE Security and Privacy Magazine*, vol. 5, no. 1, pp. 36–44, Jan. 2007, doi: 10.1109/MSP.2007.11.
- [33] “About the OWASP Foundation,” *The OWASP Foundation Inc.* <https://owasp.org/about/> (accessed Jan. 19, 2023).
- [34] “API-Security/0xa1-broken-object-level-authorization.md at master · OWASP/API-Security · GitHub,” *The OWASP Foundation Inc.*, 2019. <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa1-broken-object-level-authorization.md> (accessed Jan. 19, 2023).
- [35] “OWASP API Security Project,” *The OWASP Foundation Inc.*, 2019. <https://owasp.org/www-project-api-security/> (accessed Jan. 19, 2023).
- [36] “API-Security/0xa2-broken-user-authentication.md at master · OWASP/API-Security · GitHub,” *The OWASP Foundation Inc.*, 2019. <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa2-broken-user-authentication.md> (accessed Jan. 20, 2023).
- [37] “API-Security/0xa3-excessive-data-exposure.md at master · OWASP/API-Security,” *The OWASP Foundation Inc.*, 2019. <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa3-excessive-data-exposure.md> (accessed Mar. 01, 2023).
- [38] “API-Security/0xa4-lack-of-resources-and-rate-limiting.md at master · OWASP/API-Security · GitHub,” *The OWASP Foundation Inc.*, 2019. <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa4-lack-of-resources-and-rate-limiting.md> (accessed Jan. 21, 2023).
- [39] “API-Security/0xa5-broken-function-level-authorization.md at master · OWASP/API-Security · GitHub,” *The OWASP Foundation Inc.*, 2019. <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa5-broken-function-level-authorization.md> (accessed Jan. 21, 2023).
- [40] “API-Security/0xa6-mass-assignment.md at master · OWASP/API-Security · GitHub,” *The OWASP Foundation Inc.*, 2019. <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa6-mass-assignment.md> (accessed Jan. 22, 2023).
- [41] “API-Security/0xa7-security-misconfiguration.md at master · OWASP/API-Security · GitHub,” *The OWASP Foundation Inc.*, 2019. <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa7-security-misconfiguration.md> (accessed Jan. 22, 2023).
- [42] “API-Security/0xa8-injection.md at master · OWASP/API-Security · GitHub,” *The OWASP Foundation Inc.*, 2019. <https://github.com/OWASP/API-Security/blob/master/2019/en/src/0xa8-injection.md> (accessed Jan. 22, 2023).
- [43] “Security Content Automation Protocol | CSRC,” *NIST*, 2016. <https://csrc.nist.gov/projects/security-content-automation-protocol/> (accessed Mar. 14, 2023).

## References

---

- [44] S. Radack and R. Kuhn, “Managing security: The security content automation protocol,” *IT Prof*, vol. 13, no. 1, pp. 9–11, Jan. 2011, doi: 10.1109/MITP.2011.11.
- [45] R. Croft, D. Newlands, Z. Chen, and A. M. Babar, “An empirical study of rule-based and learning-based approaches for static application security testing,” in *International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, Oct. 2021. doi: 10.1145/3475716.3475781.
- [46] G. Hao *et al.*, “Constructing Benchmarks for Supporting Explainable Evaluations of Static Application Security Testing Tools,” in *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, IEEE, Jul. 2019, pp. 65–72. doi: 10.1109/TASE.2019.00-18.
- [47] J. Yang, L. Tan, J. Peyton, and K. A. Duer, “Towards Better Utilizing Static Application Security Testing,” in *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 51–60. doi: 10.1109/ICSE-SEIP.2019.00014.
- [48] S. B. Banks, “Dynamic software security testing,” *IEEE Security and Privacy*, vol. 4, no. 3. pp. 77–79, May 2006. doi: 10.1109/MSP.2006.64.
- [49] T. Rangnau, R. V. Buijtenen, F. Fransen, and F. Turkmen, “Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines,” in *Proceedings - 2020 IEEE 24th International Enterprise Distributed Object Computing Conference, EDOC 2020*, Institute of Electrical and Electronics Engineers Inc., Oct. 2020, pp. 145–154. doi: 10.1109/EDOC49727.2020.00026.
- [50] A. Dann, H. Plate, B. Hermann, S. E. Ponta, and E. Bodden, “Identifying Challenges for OSS Vulnerability Scanners-A Study & Test Suite,” *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3613–3625, Sep. 2022, doi: 10.1109/TSE.2021.3101739.
- [51] H. Plate, S. E. Ponta, and A. Sabetta, “Impact assessment for vulnerabilities in open-source software libraries,” in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, Sep. 2015, pp. 411–420. doi: 10.1109/ICSM.2015.7332492.
- [52] A. Rahman, C. Parnin, and L. Williams, “The Seven Sins: Security Smells in Infrastructure as Code Scripts,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, May 2019, pp. 164–175. doi: 10.1109/ICSE.2019.00033.
- [53] T. FitzMacken and D. Richards, “Azure Resource Manager overview - Azure Resource Manager | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview> (accessed Dec. 03, 2022).
- [54] M. Jacobs, J. Kulla-Mader, T. Petersen, and E. Kaim, “What is infrastructure as code (IaC)? - Azure DevOps | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code> (accessed Dec. 03, 2022).
- [55] T. FitzMacken and J. Gao, “Templates overview - Azure Resource Manager | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/overview> (accessed Dec. 03, 2022).

## References

---

- [56] J. Liebermann, E. Urban, and D. Coulter, “What is Azure DevOps? - Azure DevOps | Microsoft Docs,” *Microsoft*, 2022. <https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?toc=%2Fazure%2Fdevops%2Fget-started%2Ftoc.json&bc=%2Fazure%2Fdevops%2Fget-started%2Fbreadcrumb%2Ftoc.json&view=azure-devops> (accessed Aug. 07, 2022).
- [57] “Azure DevOps Services,” *CloudStrucc Inc*, 2022. <https://www.cloudstrucc.com/services/devops/> (accessed Aug. 08, 2022).
- [58] S. Leavitt, L. Casey, and D. Mabee, “What is Azure Boards? Tools to manage software development projects. - Azure Boards | Microsoft Docs,” *Microsoft*, 2022. <https://docs.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops> (accessed Aug. 07, 2022).
- [59] R. Bououni, K. Toliver, and G. Marlow, “Artifacts in Azure Pipelines - Azure Pipelines | Microsoft Docs,” *Microsoft*, 2022. <https://docs.microsoft.com/en-us/azure/devops/pipelines/artifacts/artifacts-overview?view=azure-devops&tabs=nuget> (accessed Aug. 07, 2022).
- [60] N. Trogh, R. Jagtap, and D. Lee, “What is Azure Test Plans? Manual, exploratory, and automated test tools. - Azure Test Plans | Microsoft Docs,” *Microsoft*, 2022. <https://docs.microsoft.com/en-us/azure/devops/test/overview?view=azure-devops> (accessed Aug. 07, 2022).
- [61] V. Machiraju, T. Sherer, and T. Petersen, “Collaborate on code - Azure Repos | Microsoft Docs,” *Microsoft*, 2022. <https://docs.microsoft.com/en-us/azure/devops/repos/get-started/what-is-repos?view=azure-devops> (accessed Aug. 07, 2022).
- [62] R. Pandey, O. Gomez, and D. Jarvis, “Azure Pipelines New User Guide - Key concepts - Azure Pipelines | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops> (accessed Nov. 07, 2022).
- [63] J. Kulla-Mader, S. Danielson, and D. Rea, “What is Azure Pipelines? - Azure Pipelines | Microsoft Docs,” *Microsoft*, 2022. <https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops> (accessed Aug. 08, 2022).
- [64] Z. Ahmed and S. Francis, “Integrating Security with DevSecOps: Techniques and Challenges,” *IEEE*, 2019.
- [65] “Introducing Snyk - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/getting-started/introducing-snyk> (accessed Jan. 25, 2023).
- [66] “Scan application code - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/scan-application-code> (accessed Feb. 11, 2023).
- [67] “Snyk Open Source - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/scan-application-code/snyk-open-source> (accessed Feb. 11, 2023).
- [68] “Snyk for .NET - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/scan-application-code/snyk-open-source/snyk-open-source-supported-languages-and-package-managers/snyk-for-.net> (accessed Feb. 18, 2023).
- [69] “SAST testing: how it works and why do you need it? | Snyk,” *Snyk*. <https://snyk.io/learn/application-security/static-application-security-testing/> (accessed Feb. 16, 2023).



## References

---

- [70] “Snyk Code - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/scan-application-code/snyk-code> (accessed Feb. 17, 2023).
- [71] “Snyk Code AI Engine - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/scan-application-code/snyk-code/introducing-snyk-code/key-features/ai-engine> (accessed Feb. 18, 2023).
- [72] “Git Security Scanning & Secrets Detection,” *GitGuardian*. <https://www.gitguardian.com/> (accessed May 13, 2023).
- [73] B. Neira, L. O’Connor, R. Lyon, and G. Sinha, “What is Azure Active Directory? - Microsoft Entra | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-what-is> (accessed Dec. 01, 2022).
- [74] C. Lin, S. Ouko, and L. Foust, “Managed identities - Azure App Service | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/azure/app-service/overview-managed-identity?tabs=portal%2Chttp> (accessed Dec. 01, 2022).
- [75] J. Gao, D. Smatlak, T. FitzMacken, and K. Sharkey, “Template structure and syntax - Azure Resource Manager | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/syntax> (accessed Jan. 14, 2023).
- [76] J. Gao, D. Smatlak, and K. Sharkey, “Parameters in templates - Azure Resource Manager | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/parameters> (accessed Jan. 17, 2023).
- [77] M. Ribbins, K. Sharkey, and J. Gao, “Create parameter file - Azure Resource Manager | Microsoft Learn,” *Microsoft*, 2022. <https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/parameter-files> (accessed Jan. 17, 2023).
- [78] R. Nair, J. Koke, and J. Erickson, “Service connections in Azure Pipelines - Azure Pipelines | Microsoft Docs,” *Microsoft*, 2022. <https://docs.microsoft.com/en-us/azure/devops/pipelines/library/service-endpoints?view=azure-devops&tabs=yaml> (accessed Aug. 11, 2022).
- [79] T. Dykstra, P. Poojari, and G. Warren, “.NET CLI | Microsoft Docs,” *Microsoft*, 2022. <https://docs.microsoft.com/en-us/dotnet/core/tools/> (accessed Aug. 11, 2022).
- [80] “The Official YAML Web Site,” *YAML*, 2021. <https://yaml.org/> (accessed Mar. 02, 2023).
- [81] “Azure App Service Management: ARM,” *Microsoft*, 2019. <https://github.com/microsoft/azure-pipelines-tasks/blob/master/Tasks/AzureAppServiceManageV0/README.md> (accessed Nov. 09, 2022).
- [82] “Use Snyk in your IDE - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/ide-tools> (accessed Feb. 18, 2023).
- [83] “Git repository integrations (SCMs) - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/integrations/git-repository-scm-integrations> (accessed Feb. 19, 2023).
- [84] “GitHub integration - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/integrations/git-repository-scm-integrations/github-integration> (accessed Feb. 19, 2023).

## References

---

- [85] “PR Checks for Snyk Code - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/scan-application-code/run-pr-checks/pr-checks-for-snyk-code> (accessed Feb. 19, 2023).
- [86] “Scan cloud deployments - Snyk User Docs,” *Snyk*. <https://docs.snyk.io/scan-cloud-deployment> (accessed Feb. 19, 2023).