Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Anniken Arildset
Celina Heimdal Brynildsen
Sebastian Hestsveen
Thea Urne

Securing the Software Development Life Cycle

Bachelor's thesis in Digital Infrastructure and Cybersecurity
Supervisor: Filip Holik

May 2023

NTNU
Norwegian University of
Science and Technology

Anniken Arildset
Celina Heimdal Brynildsen
Sebastian Hestsveen
Thea Urne

Securing the Software Development Life Cycle

Bachelor's thesis in Digital Infrastructure and Cybersecurity
Supervisor: Filip Holik
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Kunnskap for en bedre verden

Department of Information Security and Communication Technology

DCSG2900 - Bachelor Thesis Bachelor of Science in Digital Infrastructure and Cyber Security

Securing the Software Development Life Cycle

Author:

Anniken Arildset, Celina Heimdal Brynildsen, Sebastian Hestsveen, Thea Urne

May, 2023

Abstract

Title:	Securing the Software Development Life Cycle
Date:	22.05.2023
Participants:	Anniken Arildset Celina Brynildsen Sebastian Hestsveen Thea Urne
Supervisor:	Filip Holik, Visiting Researcher/Research Assistant, Department of Information Security and Communication Technology
Employer:	Astri Marie Ravnaas, Norges Bank Investment Management (NBIM)
Keywords:	AWS, DevSecOps, GitHub, Information Security, Infrastructure as Code, SDLC
Pages:	86
Attachments:	8
Availability:	Open

Abstract: This research aims to address the importance of securing systems and applications by focusing on implementing security testing and integrating tools into a development pipeline. The report provides a proof of concept for building a secure pipeline, emphasizing best practices and security tools to detect vulnerabilities early in the Software Development Life Cycle (SDLC). The findings and recommendations can contribute to the industry's understanding of securing the SDLC and mitigating threats.

Sammendrag

Tittel:	Securing the Software Development Life Cycle
Dato:	22.05.2023
Deltakere:	Anniken Arildset Celina Brynildsen Sebastian Hestsveen Thea Urne
Veileder:	Filip Holik, Forsker/Vitenskapelig assistent, Institutt for informasjonssikkerhet og kommunikasjonsteknologi
Oppdragsgiver:	Astri Marie Ravnaas, Norges Bank Investment Management (NBIM)
Nøkkelord:	AWS, DevSecOps, GitHub, Informasjonssikkerhet, Infrastruktur som kode, SDLC
Antall sider:	86
Antall vedlegg:	8
Tilgjenlighet:	Åpen

Sammendrag: Denne rapporten fokuserer på å adressere viktigheten av å sikre systemer og applikasjoner ved å fokusere på implementering av sikkerhetstesting og integrering av verktøy i en utviklingspipeline. Rapporten gir et proof of concept for å bygge en sikker pipeline, og legger vekt på beste praksis og sikkerhetsverktøy for å oppdage sårbarheter tidlig i Software Development Life Cycle (SDLC). Funnene og anbefalingene kan bidra til industriens forståelse av å sikre SDLC og redusere trusler.

Preface

We want to thank our supervisor, Filip Holik, for all help and guidance we received during the project period.

Next, we would like to thank all professors who have helped us answer questions that arose during the project.

We also want to thank family members who took the time to read our thesis and provide us with feedback.

Lastly, we thank our NBIM contact persons, Stian Hagbø Olsen, Stian Kristoffersen, and Astri Marie Ravnaas for their excellent guidance and valuable feedback. They assisted with various issues and were frequently used for guidance whenever questions or problems arose.

Table of Contents

List of Figures

List of Tables

Code Listings

Acronyms

Amazon EC2 Amazon Elastic Compute Cloud. 49, 61, 62

Amazon SNS Amazon Simple Notification Service. 64

AMIs Amazon Machine Images. 49

API Application Programming Interface. 40, 56, 71

AWS Amazon Web Services. xiv, xv, 3, 5, 23, 32–34, 44, 47–49, 51, 52, 56–58, 64, 65, 68, 69, 74, 75, 77, 82, 84, 85, 120

CI/CD Continuous integration and continuous deployment. 11

CLI Command-Line Interface. 54, 85

CVE Common Vulnerabilities and Exposures. 21, 41, 55

CVSS Common Vulnerability Scoring System. xiv, 21

CWE Common Weakness Enumeration. 21, 22, 55

DAST Dynamic Application Security Testing. xiv, 4, 5, 16, 17, 19, 21, 29, 30, 44, 62, 67, 69–71, 75, 76, 84–86, 96

DevSecOps Development, Security, and Operations. xiv, 6, 12

GPG GNU Privacy Guard. 46, 53

IaaS Infrastucture as a Service. 23

NBIM Norwegian Bank Investment Management. 2–4, 117, 118, 120, 125

NIST National Institute of Standards and Technology. 31, 36

NVD National Vulnerability Database. 21, 41

OWASP Open Worldwide Application Security Project. xvi, 20–22, 29, 44, 52, 61, 62, 69–71, 96

PaaS Platform as a Service. 23

SAST Static Application Security Testing. xiv, 4, 5, 15–17, 19, 21, 27–29, 39, 69–71, 75, 76, 84–86

SCA Software Composition Analysis. xiv, 4, 5, 17–19, 21, 28, 29, 41, 69, 70, 75, 76, 84–86

SDLC Software Development Life Cycle. xiv, 2–7, 10, 11, 13–18, 20, 26, 27, 33, 35, 39, 67, 74, 84–86, 117, 118, 120

SLSA Supply-chain Levels for Software Artifacts. xiv, 35, 36, 73, 76

SSDF Secure software development framework. 36, 74, 76

SSH Secure Shell. 46, 53

VPC Virtual Private Cloud. 62

YAML Yet Another Markup Language. 54

ZAP Zed Attack Proxy. xvi, 29, 44, 61, 62, 69, 71, 96

Glossary

Appspec An Application specification file, or appspec file, is a YAML or JSON formatted file used by CodeDeploy containing instructions on how to deploy and configure the application. 61

Artifact An artifact is any software package that is used as a building block for applications. 28, 49, 57

Buffer-overflow Buffer overflow happens when the size of data surpasses the storage capacity of a memory buffer, leading to potential system vulnerabilities. 15

Buildspec Buildsec is a collection of build commands and related settings in a YAML format that CodeBuild uses to run a build. 48, 59

Compute platform The environment where software is executed. This could be either the operating system or physical hardware. 49, 60

Cross-site scripting Cross-site scripting attacks, commonly referred to as XSS attacks, represent a form of injection where malevolent scripts are inserted into websites that are otherwise deemed secure and trustworthy. 15, 29, 33, 52

DDoS attack A DDoS (Distributed Denial of Service) attack floods a target website or online service with traffic from multiple systems, overwhelming its capacity and rendering it unavailable to legitimate users. 33

Denial-of-service attack describes the ultimate goal of a class of cyber attacks designed to render a service inaccessible. The DoS attacks that most people have heard about are those launched against high profile websites, since these are frequently reported by the media. 27

Dependencies A dependency refers to an external software component that an application relies on to ensure its proper functioning. 17, 27, 28

Front-end The frontend is everything a user sees and interacts with when they click on a link or type in a web address. The web address is also known as at URL, or Uniform Resource Locator, and it tells what webpage should load and appear in the browser. 52

GUI Stands for: Graphical User Interface and it is a graphics-based operating system interface that uses icons, menus and a mouse to navigate around. 5, 53

Idempotent Idempotence, in programming and mathematics, is a property of some operations such that no matter how many times you execute them, you achieve the same result. 72

Infrastructure as Code Infrastructure as Code (IaC) is the practice of automating the provisioning and management of IT infrastructure using a high-level programming language or configuration files. 5, 51, 71, 72, 77

Man-in-the-middle attack A man-in-the-middle (MiTM) attack is a type of cyber attack in which the attacker secretly intercepts and relays messages between two parties who believe they are communicating directly with each other. 28

Pipeline A pipeline is a systematic process that guides the development of software by seamlessly progressing through the stages of building, testing, and deploying code. xiv, 11, 16, 26, 27, 30, 39, 47, 48, 51, 56, 57, 69, 72, 74, 76, 77, 82, 84–86

Provenance Set of metadata providing information about how the outputs were generated, which includes identifying the platform used and any external parameters involved in the process. 36, 73

Shift-left Shift left is the practice of performing testing, quality assurance, and performance evaluation early in the development process, typically prior to coding. 3

SQL-injection SQL injections are a type of web security vulnerability that enables attackers to manipulate the queries executed by an application on its database. 15, 29, 33, 52

Chapter 1

Introduction

1.1 Background

Norwegian Bank Investment Management, from now on referred to as NBIM, is a division within the central bank responsible for overseeing the Government Pension Fund of Norway, which has a worth of 14,000 billion Norwegian kroner [1]. Due to its significant value, the fund is a major target for potential malicious actors. It faces an average of three severe cyber attacks daily, totalling around 100,000 attacks each year. Out of these, more than 1,000 are considered significant threats [2]. Therefore, it is crucial for NBIM, as well as other organizations, to ensure the security of their systems and applications before deploying them into their cloud services.

Software Development Life Cycle (SDLC) describes how software applications are built - from planning through implementation and running in production. It also includes ensuring security at the different stages of software development. In order to accommodate frequent deployments to production, it is essential to automate the security testing by building it into the deployment pipeline. Security testing can further benefit from shift-left, where testing is done as early as possible in the pipeline. Implementing a strong and secure software development life cycle is essential to prevent attacks from hackers and other malicious actors on the application.

Securing the SDLC is a large and actively developed area with much industry interest. Demonstrating the integration and practical application of various tools and methods can benefit both NBIM and other organizations.

1.1.1 Problem area

The technology industry is in constant development. With this development comes a rapidly expanding threat landscape. How developers approach IT and security must accompany this rapid development to secure systems and applications from malicious actors. Securing the Software Development Life Cycle has become a common topic in the tech industry, and many resources are available to help organizations implement best practices and adequate security measures. However, finding the appropriate resources can take time, and automating the complete distribution process and automated testing using multiple tools can be time-consuming. Therefore, NBIM is looking for proof of concepts for building a secure pipeline using best practices and implementing multiple security tools to scan for security misconfigurations and vulnerabilities at crucial pipeline stages.

1.2 Scope limitations

The thesis covers all phases of the Software Development Life Cycle, and gives an overview of each phase and the importance of securing them. The phases consist of planning, implementation, testing, deployment, and maintenance. However, due to the scope, only the last four phases will be the main priority for testing purposes and building a secure pipeline from GitHub to AWS. The group has also decided not to focus on shift-left testing within the life cycle, as the focus is on the later phases.

As a part of the thesis, the group utilized AWS as it was required to use. However, considering the vastness of the AWS platform, the group chose not to explore the tools available within AWS extensively. Instead, the group opted to use the tools necessary at that time and were relatively simple to implement rather than focusing too much on the tools one could use to build the pipeline.

1.3 Target group

The thesis has multiple target groups. The primary target group is NBIM, the stakeholder for this thesis, for whom the group will produce a comprehensive report on the task assigned to them. However, this report has the potential to benefit other organizations, and therefore, another target group is any organization that utilizes the SDLC and aims to improve its security.

1.4 Goals

1.4.1 Performance goals

- P1: Collaborate effectively with team members to ensure the timely completion of tasks.
- P2: Successfully integrating security tools (e.g., SAST, DAST, SCA) into the SDLC pipeline.
- P3: Implement an automated pipeline using Terraform to build, test, and deploy applications.

1.4.2 Result goals

- R1: Develop a secure and automated pipeline for the SDLC process using Terraform.
- R2: Produce a report summarizing the project results and recommendations for improving the SDLC pipeline security.

1.5 The group's academic background

The group is in the third and final year of a bachelor's degree program in Digital Infrastructure and Cyber Security at NTNU Gjøvik. Throughout the studies, the group has covered various courses such as risk management, ethical hacking, cyber security and teamwork. These subjects have equipped the group with relevant knowledge for their thesis work.

1.5.1 Knowledge that had to be acquired

The group had to acquire various new aspects of software development for the thesis, as it was not covered extensively in their studies. These aspects include topics like SDLC, Static Application Security Testing, Dynamic Application Security Testing and Software Composition Analysis, and more. Due to the inclusion of the practical parts in the thesis, like testing insecure code, the group had to learn about building a pipeline and integrating various security tools to ensure a secure development. As a significant part of the main scope focused on tools integrated into GitHub and Amazon Web Services, the group had to acquaint themselves with these tools and the various features of GitHub. Despite the group's experience with GitHub from previous courses, numerous features were still unfamiliar. In contrast, AWS was unfamiliar, and the group had no prior knowledge. However, the group had used similar cloud services like Microsoft Azure in previous courses.

In addition, the group had to familiarize themselves with Terraform and use it to establish the automated pipeline. Using Terraform instead of working solely in the GUI was based on several reasons to use infrastructure as code over manual configuration. For instance, automation could enhance efficiency and decrease errors in the development process.

1.5.1.1 Why this task was chosen

The group chose this task because of the shared interest in the Software Development Life Cycle. SDLC is complex and contains multiple stages, and ensuring its security can be considered important in today's digital landscape. By looking at tools integrated in Github and AWS, the group aimed to understand better how to use these tools to identify and mitigate potential security risks at the different stages of the SDLC.

1.6 Framework

1.6.1 Timeframe

The task completion deadline runs from the 11th of January, 2023, to the 22nd of May, 2023. It was agreed early on that the first draft would be submitted to the supervisor 3rd of April to allow the supervisor enough time to read through the thesis and give feedback. Then, the group decided to submit the final draft to the supervisor on the 1st of May, which would give a month between the first draft and the final draft to work.

1.6.2 Other

The stakeholder requested a comprehensive report that did not specifically focus on their systems, as they were unable to provide access to their systems or development environment for the group. Therefore, the group had a significant amount of freedom in determining the scope and specific limitations of the thesis.

1.7 Methodology

The group adopted a DevSecOps approach as a working method during the project, which includes incorporating security into the DevOps process at all stages of the SDLC. Despite focusing solely on the last four phases of the SDLC, the team maintained a DevSecOps mindset, viewing security as an essential component of the entire process rather than a separate phase.

1.8 Research methods

The group used different methods to gain knowledge that was needed to fulfill the requirements that were given. Below are some of the research methods used to gain knowledge about the topic.

1.8.1 Interviews/meetings

At the beginning of the project, the group conducted interviews with various lecturers at NTNU Gjøvik to collect information on relevant subjects. By getting insights from software security experts, the group acquired a fresh perspective on the topic, supplementing the knowledge gained through literature studies. Additionally, the group prepared a list of pre-determined questions to address any uncertainties about specific areas that remained unclear based on prior research.

1.8.2 Literature study

The literature study was done by researching online for relevant information about the topics, which helped the group add suitable material for the theory chapter. In addition, the study provided the essential knowledge required to address secure SDLC practices, which formed the basis for delivering appropriate recommendations.

1.9 Software utilized for writing

This section contains an overview of software utilized in the process of writing the thesis.

- **ChatGPT:** Primarily used for rephrasing when uncertain how to formulate the message.
- **Grammarly Premium:** For correct grammar, consistency and precise wording.
- **Overleaf:** Cooperative LaTeX editor used for writing the thesis.

1.10 GitHub organization

As part of the thesis, the group created a GitHub organization containing different repositories, including all the code utilized for creating an automated pipeline and various security tools and a backup of our thesis.

https://github.com/orgs/DCSG2900-Bachelor-thesis/repositories

1.11 Thesis structure

When reading the thesis, some things can be helpful to note. First, there are clickable links that make the navigation to chapters, sections, acronyms, figures, tables, and sources go as seamlessly as possible. The language used throughout the thesis is English. The same goes for all meeting notes and other relevant appendixes.

1.11.1 Chapters

- **Chapter 1 - (Introduction)**: Contains an introduction to the thesis.
- **Chapter 2 - (Theory)**: Contains theory that the group considers important to have some knowledge about to understand the thesis as a whole.
- **Chapter 3 - (Pipeline security)**: Contains an overview of what to implement to secure data sent through the pipeline and what to include to secure the pipeline.
- **Chapter 4 - (Analysis of security tools for the pipeline)**: Contains an overview of the different tools the group has implemented into their pipeline, with an analysis of these tools.
- **Chapter 5 - (End-to-end pipeline)**: Contains the steps done for pipeline automation and implementation of chosen security tools.
- **Chapter 6 - (Discussion)**: Contains an in-depth discussion of choices made throughout the thesis.
- **Chapter 7 - (Conclusion)**: Contains an overview of the work process and a conclusion to the group's findings.

Chapter 2

Theory

2.1 Introduction

This chapter provides an explanation of different types of concepts that is essential to understand before reading the rest of the thesis. It covers various forms of software testing, multiple techniques of security testing, and other relevant information deemed necessary to understand the thesis.

2.2 Software Development Life Cycle

Software Development Life Cycle can be defined as *”structured process that enables the production of high-quality, low-cost software, in shortest possible production time. The goal of the SDLC is to produce superior software that meets and exceeds all customer expectations and demands”*[3]. SDLC consists of several phases providing a software development testing and deployment framework. The Software Development Life Cycle is an iterative process, where each life cycle phase can undergo multiple iterations and be repeated, with each iteration building upon the previous one [4]. This iterative method allows continuous improvement and refinement, producing a final product that meets or exceeds customer needs and expectations. Using this approach, software development teams may ensure that their product is of the greatest quality while still being completed in a reasonable time and cost. Below are the five phases that fulfill the SDLC.

2.2.1 Planning

Planning is the first phase of the SDLC [5]. During this phase, the team determines the project's goals and objectives. The team should discuss resources, costs, and time and create a timeline for the work during this phase. Additionally, the team should develop a project plan where they identify, prioritize, and assign tasks and resources necessary to construct the structure. This stage is repeated when changes or new features are presented [6]. In a CI/CD pipeline, the team receives feedback and includes it in a new planning phase.

2.2.2 Implementation

In the implementation phase, the actual development of the software takes place [7]. The software design is translated into code using programming languages. Once the team completes and reviews the code, they initiate the build process and conduct several security scans. The implementation phase is considered important since this is the phase that involves the actual development of the software and the preparation of the software for deployment into the environment.

2.2.3 Testing

In the testing phase, the development team verifies that the software fulfils functional, performance, security, and quality criteria established in earlier stages of the SDLC [8]. The software undergoes both manual user testing and automated testing to detect and correct possible defects. During testing, the development team confirms that the software fits the user's requirements.

2.2.4 Deployment

In the deployment phase, releasing the software into the environment can be considered the main priority [9]. During this phase, the deployment team works with the development team and stakeholders to arrange the software's release to guarantee a seamless process. The plan also includes creating the deployment timeline, selecting a method for the deployment, and defining the software's environment. Initially, the deployment procedure of an application may take a long time. However, after the application has been deployed for the first time, future changes may be delivered more quickly since the same configuration is reused, resulting in significantly reduced deployment time.

2.2.5 Maintenance

In the maintenance phase, the software is monitored to ensure it functions according to its intended design [10]. The team does any refactoring and upgrades if it is needed. Software monitoring can be done differently depending on how the maintenance phase has been set up. However, the most common way of monitoring is usually through real-time reporting or ad-hoc reporting systems

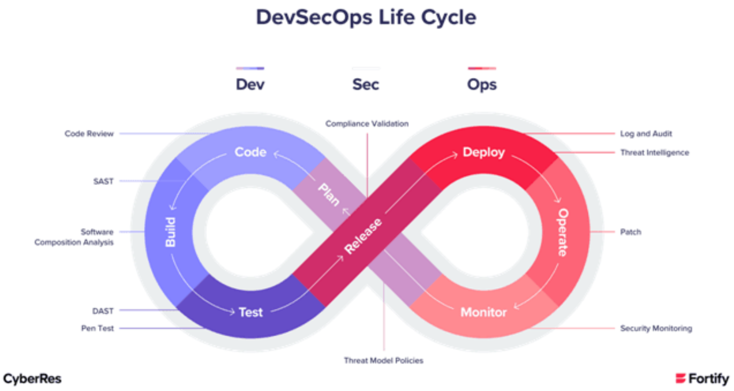


Figure 2.1: DevSecOps Life Cycle
[11]

2.3 Functional testing vs. security testing

Software testing is an extensive area, and each test varies according to its purpose or process. The primary goal of software testing is to verify the quality of software systems through planned and structured testing under controlled conditions [12]. Functional testing emphasizes the software's behavior and that the software is working as expected. The functional test cases are based on the software requirements specified by the stakeholder. Several functional testing methodologies, such as unit and integration testing, can be conducted.

Unit testing involves testing small code components, known as units, to determine if the units behave as expected[13]. These tests are typically executed early in the SDLC to identify bugs early on and save time in the rest of the process. The unit testing aims to run tests on all possible components in an isolated test environment to confirm if the code operates as anticipated. On the other hand, integration testing evaluates how the previously tested components perform when integrated into a more extensive system and communicate across various components.

Security testing wants to break the software to uncover vulnerabilities and verify its security features [14]. The testing aims to identify all possible weaknesses that attackers might exploit. The tests are performed from the attacker's point of view. These tests can be done manually or by software tools called automated security testing tools. The goal of evaluating security functionalities is to verify if protective measures like authentication are functioning as expected. Security testing will also try to simulate attacks on the software and determine its capability to defend against them. White box testing, black box testing, and grey box testing are some examples of security testing methods. This topic is described more in section 2.4.1.

2.4 Application security testing

Application Security Testing can be described as a *"process of making applications more resistant to security threats by identifying security weaknesses and vulnerabilities in source code"* [15].

2.4.1 Box testing

Below are some of the different security tools that can be used to make applications more resistant to security threats and which will be used in securing the SDLC. The various testing tools utilize a type of testing technique called "box testing".

2.4.1.1 Black box testing

Black box testing is a testing technique that primarily focuses on the functionality and behavior of an application without any knowledge of its internal structure and processes [16]. Therefore, the tester can treat the application as a black box, where only the inputs and outputs are visible, and the internal workings remain unknown. The technique makes it a practical approach to test the functionality of an application, as the tester verifies whether the input produces the expected output without any knowledge of the underlying code or design. As a result, black box testing is often used as a form of functionality testing [17]. If the software produces the expected output for the given input, the tester considers it to have passed the black box test.

2.4.1.2 White box testing

White box testing focuses on the application from within. During such tests, the tester will examine the source code and infrastructure [18]. The testing covers paths, statements, and branches, among other things. A white box tester looks for security holes by testing the code and therefore requires to know programming and IT. The tester performs security testing, like testing for memory leaks, and functional testing, like unit tests. White box testing can be pretty complex, and when running such tests on a large amount of code, it can take days or even weeks to test.

2.4.1.3 Grey box testing

Grey box testing is a method that limits the user's knowledge of the different components being tested [19]. It combines white box testing and black box testing, where the user can access internal code or design without enough access to run a full white box test.

2.4.2 SAST

Static Application Security Testing (SAST) is a type of white-box testing that analyzes the source code of an application to identify security vulnerabilities within the code [20]. This testing method usually occurs during the development phase of the Software Development Life Cycle. The primary purpose of this method is to identify and remediate security issues before the application is deployed.

SAST tools scan source code for known security threats, such as cross-site scripting, SQL-injection, and buffer-overflow. SAST tools also warn about any security weaknesses that may lie in the code that can potentially be exploited. After the tool has gone through the code, it generates a report that contains the different vulnerabilities that it has identified. In addition, the report includes a more in-depth description of the vulnerability and remediation on how to fix it.

One of the advantages of SAST is that it gives detailed information about the source of the vulnerability, which gives the developer a better understanding of how to fix the issue.

There are, however, some limitations to SAST tools. For example, it can only detect vulnerabilities in the source code, meaning it cannot detect vulnerabilities that result from the interaction between different components of an application. In addition, SAST tools often generate numerous false positives since they analyze source code without considering the entire code context [21].

SAST scans should be implemented as early in the SDLC as possible to prevent vulnerabilities from being present throughout the pipeline [22]. Figure 2.1 suggests that the SAST scan should be done before building the code in the implementation phase. It is also possible to run this test during the testing phase, as shown in Figure 2.2.

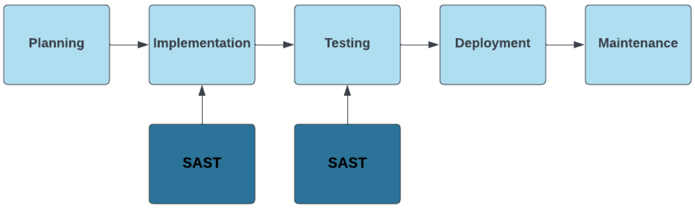


Figure 2.2: Where SAST can be performed in SDLC

2.4.3 DAST

Dynamic Application Security Testing (DAST) uses a black box testing technique that evaluates the security of an application by performing security assessments of a running instance of the application [23]. Unlike SAST, which analyzes the source code of an application, DAST evaluates the application as its being used. The evaluation includes the interaction of different components and the runtime environment.

DAST simulates real-world attacks, which is done by sending malicious requests and inputs to the application it is testing and then monitoring the responses. In the end, the tool generates a report that includes the identified vulnerabilities, including a more in-depth description and remediation on how to fix the issue.

An advantage with DAST is that it can identify security issues that are not detectable through SAST. These security issues can, for example, be interactions between different components. Another advantage is that with DAST, it can identify different vulnerabilities that get triggered, for example, when the application is under heavy load or when specific inputs are received.

However, there are some limitations with DAST as well. One is that it can only detect vulnerabilities present in the deployed version of the application and cannot give an in-depth description of vulnerabilities in the source code. Additionally, DAST may generate false positive. Though the tools generates less false positives compared to SAST, eliminating these alerts require more advanced analysis and verification technologies [24].

In Figure 2.3, DAST is suggested to perform in the testing and maintenance phase since the scan is done on a running application and is, therefore, past the implementation stage. The test should be done before deployment to prevent a vulnerable application from going public. Scans should also be done after it is deployed [25], in the maintenance phase, since vulnerabilities may occur after deployment and must be identified to prevent potential problems.

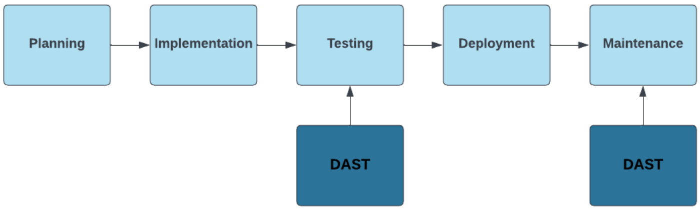


Figure 2.3: Where DAST can be performed in SDLC

2.4.4 SCA

Software Composition Analysis (SCA) is a type of grey box testing that analyzes the dependencies of a software application to identify and manage potential security risks [26]. The main objective of the SCA is to identify third-party components that may contain security vulnerabilities.

SCA scans the application's code to identify all of its dependencies, including the different versions of the components used. It then cross-references these dependencies to different databases with known vulnerabilities and generates a report containing any potential risk. Compared to the other tests, the report also includes an in-depth description of the vulnerability and a recommendation to update the components to newer versions or replace these.

An advantage with SCA is that it can quickly identify risks that may be introduced from third-party components. It is relatively common that modern applications rely on many different dependencies, making SCA useful. It can provide a comprehensive view of the security risks associated with an application and help developers make informed decisions about the security of the developed applications.

However, SCA may give some false positives that can occur because the developer has an extensive library in the code. In addition, it can trigger because of a dependency that is never used and is, therefore, impossible to exploit.

SCA scans are done at an early stage in modern software development [26]. Shown in Figure 2.1 it is performed in the build stage. Therefore, Figure 2.4 suggests SCA scans to be performed in the implementation phase. The scan is also suggested in the testing and maintenance phases. The testing phase is an alternative or an addition to the implementation phase. The maintenance phase should include checks, as dependencies may become vulnerable after deployment.

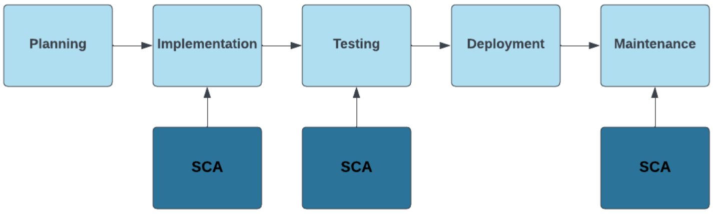
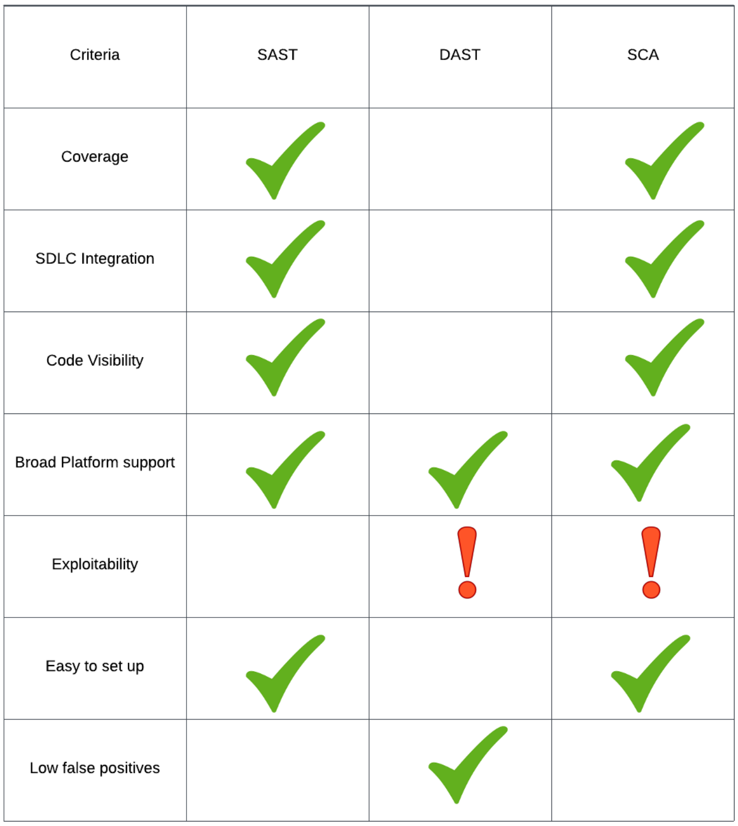


Figure 2.4: Where SCA can be performed in SDLC

2.4.5 Comparison of SAST, DAST, and SCA

Upon reviewing the different security application tests, similarities between SAST, DAST, and SCA were discovered. Table 2.5 shows commonalities and differences, which indicates that a combination of the different testing mechanisms can enhance the testing process of the application.



Criteria	SAST	DAST	SCA
Coverage	✓		✓
SDLC Integration	✓		✓
Code Visibility	✓		✓
Broad Platform support	✓	✓	✓
Exploitability		!	!
Easy to set up	✓		✓
Low false positives		✓	

Note: The distinction between the exclamation marks and the checkmarks lies in the meaning of them. The exclamation marks indicate that the criteria are considered negative, while the checkmarks signify that the criteria are positive.

Figure 2.5: Comparison of SCA, DAST, and SAST
Adapted from: [27]

2.5 The significance of software security testing

Security testing plays a critical role in the Software Development Life Cycle, as it is employed to identify potential security vulnerabilities in the system and prevent real-world attacks [14]. It is a process where the system's security is evaluated, and its possible security weaknesses and vulnerability risks are identified.

According to IBM's report, in 2022, the cost of a data breach was estimated to be USD 4.35 million [28]. Investing in cyber security measures can save money for a company in the event of a cyber attack. The report states that companies that have implemented zero trust security measures saved about USD 1 million in breach costs on average compared to those that have not. To repair a vulnerability in the planning phase costs an average of USD 500 [29]. Starting software testing early in the SDLC reduces costs and saves time. As mentioned in section 2.3, various types of testing should be executed on an application, including testing of both the written code and the libraries that are integrated and being used.

2.6 OWASP Top 10

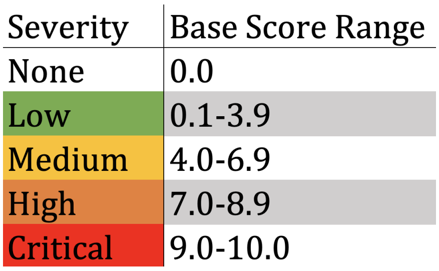
Open Worldwide Application Security Project (OWASP) serves as a standard reference document for web application security and for developers to raise their knowledge of potential security threats. The document contains a prioritized and exemplified list with recommendations for fixing ten critical security flaws commonly seen in web applications. The function of this document is to educate readers on the most common security risks that may occur. Developers and security professionals may use this knowledge and implement it into their security policies, reducing the frequency of these threats in their applications.

2.7 Vulnerability risk rating

Discovered vulnerabilities can be rated by standardized systems like CVSS, CVE, CWE, and OWASP Risk Rating Methodology. These databases are frequently used in application security testing, particularly in SCA scans, to uncover code patterns that may indicate a common vulnerability within the code. While they may also be utilized in DAST and SAST scans, they are most commonly used in SCA scans.

2.7.1 Common Vulnerability Scoring System (CVSS)

Common Vulnerability Scoring System, known as CVSS, is *”a Security Content Automation Protocol (SCAP) specification for communicating the characteristics of vulnerabilities and measuring their relative severity”*[30]. The system gives vulnerabilities a numeric score based on their severity. The scores can be translated into low, medium, high, and critical to assist organizations in evaluating and ranking their vulnerabilities. CVSS is currently at version 3.1 [31].



Severity	Base Score Range
None	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Figure 2.6: CVSS v3 Ratings
Adapted from: [32]

2.7.2 Common Vulnerabilities and Exposures (CVE)

Common Vulnerabilities and Exposures[1], known as CVE, is, according to their website,*”a list of records each containing an identification number, a description, and at least one public reference for publicly known cyber security vulnerabilities”*[33]. The CVE program aims to determine, describe, and categorize cybersecurity vulnerabilities made public. All discovered vulnerabilities will be put into records and sent to National Vulnerability Database (NVD)[2], which is a list of vulnerabilities maintained by the United States government.

[1] Available at: https://www.cve.org/
[2] Available at: https://nvd.nist.gov

2.7.3 Common Weakness Enumeration (CWE)

Common Weakness Enumeration[3], known as CWE, is *"a community-developed list of common software and hardware weakness types that have security ramifications"*[34]. It serves as a consistent benchmark for security solutions that address vulnerabilities and as a baseline for identifying, mitigating, and preventing weaknesses. CWE 's goal is to provide instructions for those who have control over and maintain source code to stop the vulnerabilities at the source.

2.7.4 OWASP Risk Rating Methodology

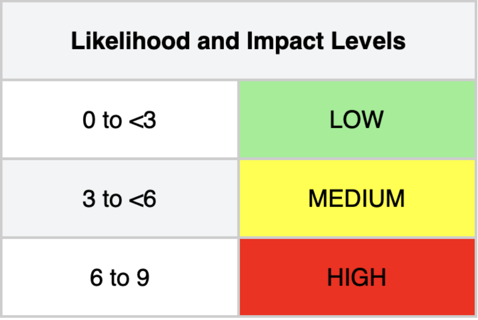
OWASP Risk Rating Methodology contains a formula that calculates a risk score for each vulnerability based on likelihood and impact [35]. Multiple factors make up both likelihood and impact.

Factors composing the estimation of likelihood are separated into different groups related to the threat actor and the vulnerability. The set of factors related to the threat actor is skill level, motive, opportunity, and size. The factors related to the vulnerability are the ease of discovery, ease of exploit, awareness, and intrusion detection. Each factor will have different options and receive a likelihood rating from 0-9.

Factors composing the estimation of impact are divided into *technical* and *business* impacts. The technical impact is broken down into confidentiality, integrity, availability, and accountability. Further, the business impact is divided into financial damage, reputation damage, non-compliance, and privacy violation. All of the factors will receive an impact rating from 0-9.

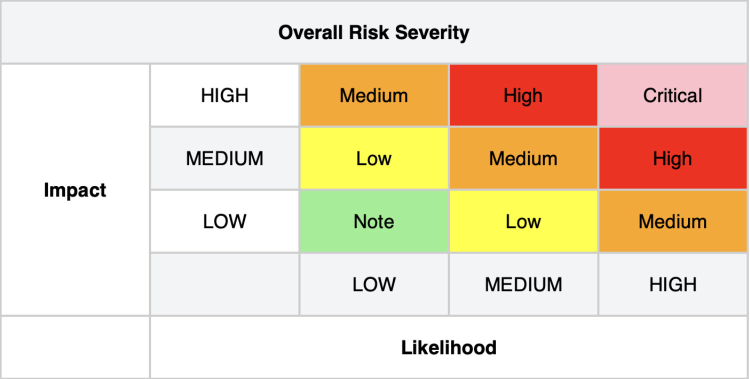
Combining the likelihood and impact ratings produces an overall risk severity level, which can be classified as low, medium, or high, as shown in figure 2.7, to determine its severity. These levels can be further combined to determine the final severity of the risk, as illustrated in figure 2.8.

[3] Available at: https://cwe.mitre.org/



Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

Figure 2.7: The likelihood and impact levels



Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Figure 2.8: Severity level based on impact and likelihood

2.8 Amazon Web Services

Amazon Web Services[4], known as AWS, is a cloud computing platform offered by Amazon [36]. It offers a range of services from Infrastucture as a Service (IaaS) to Platform as a Service (PaaS), which allows users to run their applications and keep their data in the cloud. By utilising the pay-as-you-go cloud computing model, customers can easily adjust their resources without investing in physical infrastructure.

[4] Available at: https://aws.amazon.com/

2.9 GitHub

GitHub[5] is a web-based software development platform where developers can collaborate and store open and closed-source projects [37]. The developers can manage their projects in repositories and track bugs and issues. It offers collaboration tools, including pull requests, code reviews, and project management functionalities. Git is a version control system utilized to manage and monitor file versioning. GitHub employs this technology as the basis for its service. The version control allows developers to work on code simultaneously, track changes, and merge contributions from multiple contributors.

[5] Available at: https://github.com/

Chapter 3

Pipeline security

3.1 Introduction

pipeline security is a large part of securing software development, from source code to production deployment. Pipeline security is categorized into two aspects - security in the pipeline and security of the pipeline [38]. In many ways implementing both security in the pipeline and of the pipeline will increase the overall security. Further, the group looks closer into essential security measures within these two.

Figure 3.1 displays a pipeline without any security measures implemented. The initial four stages in the pipeline are considered the implementation phase in the SDLC. Furthermore, the security testing stage in the pipeline is considered the testing phase, the deployment stage is the deployment phase, and so on. The following chapter presents the different measures that is added to the pipeline. In the end, a complete and secure pipeline is achieved.

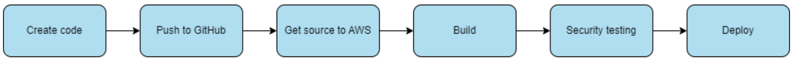


Figure 3.1: Pipeline without any security measures

3.2 Security in the pipeline

Security in the pipeline ensures the code's utmost safety from various security threats. The pipeline actively incorporates security measures and implements strict controls. The pipeline commonly consists of multiple phases like code development, testing, and deployment, all of which may be susceptible to various security threats, like unauthorized access, data breaches, malware, and denial-of-service attacks. Therefore, security in the pipeline is crucial to secure the integrity and confidentiality of software applications and data. The pipeline can use the following tools below to enhance its security. For examples of the specific tools, see chapter 4.

3.2.1 Code scanning

According to Microsoft's best practices for secure SDLC [39], a SAST tool should be included in the pipeline. A SAST tool can be, for example, code scanning. Code scanning is a security measure that analyses code with the help of a tool to find security vulnerabilities and coding errors. Code scanning serves as a preventive measure against developers introducing new issues. In Figure 3.2, the SAST scan is set to be in GitHub [20]. By doing this scan there, it is done at the earliest stage possible, before the code is moved forward in the pipeline, eliminating vulnerabilities as soon as possible.

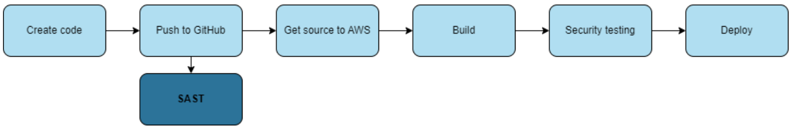


Figure 3.2: Pipeline with implemented SAST scan

3.2.2 Scan dependencies and open source libraries

Dependencies can be divided into two parts: *direct* and *transitive* [40]. A direct dependency is a directly referenced software component in an application. A transitive is a functional software component necessary for an application's direct dependencies. These dependencies may have their own set of direct and indirect dependencies, resulting in a recursive tree of transitive dependencies affecting the application. This means that the dependencies used in the code may be linked to numerous additional dependents, creating a large supply chain. To secure these supply chains, in addition to vulnerability scanning, the company can create a clear policy for evaluating and managing dependencies, including criteria for selecting secure and trustworthy libraries and frameworks. They should also limit unnecessary or outdated dependencies, as these can increase the attack surface and create unnecessary risk.

All dependencies, open-source libraries, and third-party artifacts that have been utilized should be validated [41]. To validate a file's integrity, compare the signature of an artifact to the signature generated by the artifact provider. This comparison helps detect any unauthorized alterations, tampering, or corruption of dependencies that may occur due to a man-in-the-middle attack or a compromise of the artifact repository. If any third-party software is used in the application, conducting an SCA scan with appropriate tools is crucial to detect any vulnerable open-source software implemented.

The SCA is used in conjunction with the SAST tool in Figure 3.3. Similarly to SAST, it is done in GitHub to patch the vulnerable dependencies early in the implementation phase [26].

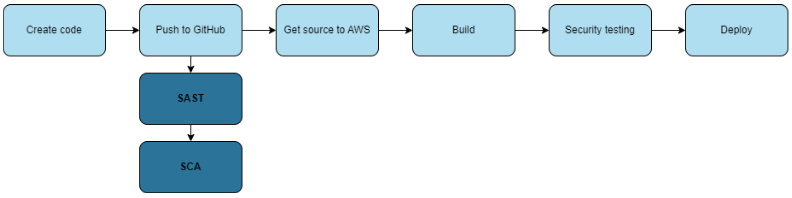


Figure 3.3: Pipeline with implemented SCA scan

3.2.3 Secret scanning

To prevent or identify accidental exposure of "secrets", like access tokens, SSH keys, or other credentials, secret scanning should be executed on the repository where the source code is stored [42]. Such secrets can give unwanted access to, for example, accounts, software, or cloud providers. With access to secrets like cloud credentials, a threat actor could, among other harmful attacks, scale up the use of various costly resources, costing a company much more than what they have budgeted for [43]. According to GitGuardian's early report on secret leaks [44], they detected 10 million secrets leaked in 2022, which is an increase of 67% from 2021. According to the report, 1 out of every 10 GitHub authors has accidentally shared a secret in their repository, highlighting a growing problem with leaked confidential information. Developers can use secret scanning tools to search for vulnerabilities and receive alerts regarding potential security risks.

In Figure 3.4, it is recommended to perform secret scanning together with SAST and SCA for the same reason, explained in sections 3.2.1 and 3.2.2.

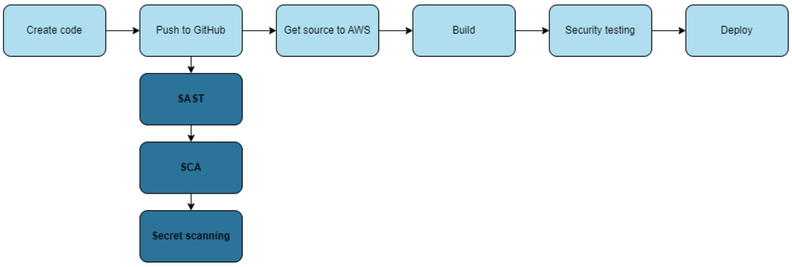


Figure 3.4: Pipeline with implemented secret scan

3.2.4 Dynamic scanning

In software best practices, it is recommended to run multiple tests and scans to identify bugs and errors - where one of these tests is Dynamic Application Security Testing (DAST) [41]. This scanning method tries to penetrate the application, attempting to identify its vulnerabilities and weaknesses. A specialized tool for DAST scan, such as OWASP ZAP, can be implemented to identify security risks like cross-site scripting, SQL-injection or path traversal [45].

3.2.5 Manual security testing

Even though DAST can be used to identify potential vulnerabilities, certain types of threats may go undetected [46]. For this reason, the company should engage a red team, a group of experts capable of performing penetration testing (pentesting). A penetration test will provide a more realistic test, as it simulates a real-world attack, detects more complex vulnerabilities, and provide a more comprehensive view of an application's security posture. In addition, a penetration test can verify the results of a DAST scan by assessing whether a vulnerability can be exploited and the extent of the damage it could cause.

Figure 3.5 shows that the DAST scan and penetration test are positioned later in the pipeline because they require a running application to be executed [25]. These tests cannot be performed earlier but must be conducted before the application is deployed.

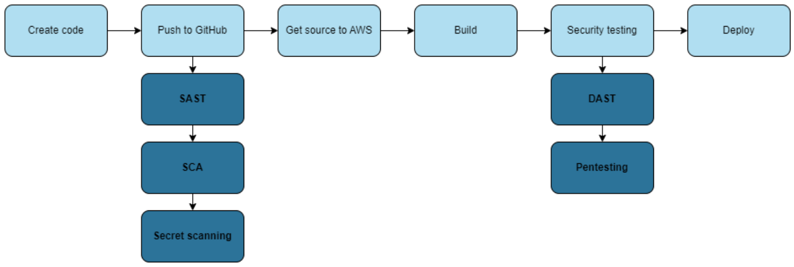


Figure 3.5: Pipeline with implemented DAST scan and pentesting

3.3 Security of the pipeline

Security of the pipeline refers to the measures taken to protect the pipeline and the underlying infrastructure and network involved in processing the code that passes through it. To ensure the pipeline 's security, it is limited to performing its intended functions and stops unauthorized access to restricted resources.

3.3.1 Branch Protection

Branch protection ensures that specific criteria are met before code can be merged [47]. This feature allows users to create branch protection rules that enforce specific workflows for one or multiple branches, such as mandating an approving review or passing status checks for all pull requests merged into the protected branch. Access to this feature of GitHub is available for all users.

Enforcing branch protection makes introducing errors and vulnerabilities into the secured branch less likely. In addition, branch protection creates a more precise development process by providing guidelines and requirements for making changes in the code. This helps ensure that all team members are aligned towards the same goal and working together effectively.

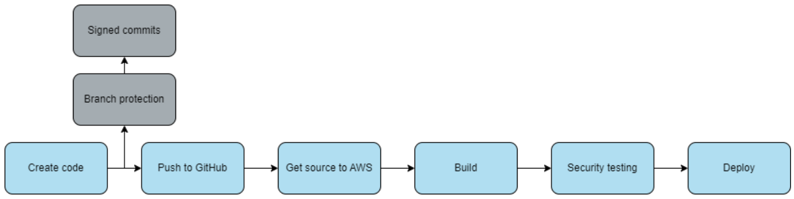


Figure 3.6: Pipeline with implemented branch protection rules

3.3.2 Access Control

Access control is crucial to regulating individuals' access to specific resources such as GitHub [48]. It involves implementing measures to determine the appropriate level of access for each individual while following the "least privilege" principle. As specified by NIST [49], this principle entails designing a security architecture, granting each entity only the minimum system resources and authorizations needed to perform its function.

In GitHub, permissions control access, which refers to the capability to execute specific tasks. In addition, team members can have specific roles assigned to them and specific permissions can be granted to individuals and groups. By following these measures, organizations can effectively manage access control and reduce risks associated with unauthorized access to critical resources.

Secure authentication is a crucial aspect of maintaining system security [50]. It involves implementing measures to ensure that only authorized users can access a system and that their access is limited to the specific resources needed to perform their duties. Conditional access policies can help organizations improve their authentication process [51]. By specifying specific rules and conditions, the system can determine the appropriate times, locations, and methods for users to gain access. For example, an organization might require multi-factor authentication for all users accessing the system outside the corporate network. In addition, they might limit access to specific applications or data based on the user's role or location.

Conditional access is an essential part of an organization's overall security plan. It provides effective system access management, lowering the risk of unwanted access or data breaches and ensuring regulatory compliance. Furthermore, applying this access control to all system components at every pipeline stage is advisable for optimal security, as shown in Figure 3.7. As a result, necessary and consistent access control implementation is crucial.

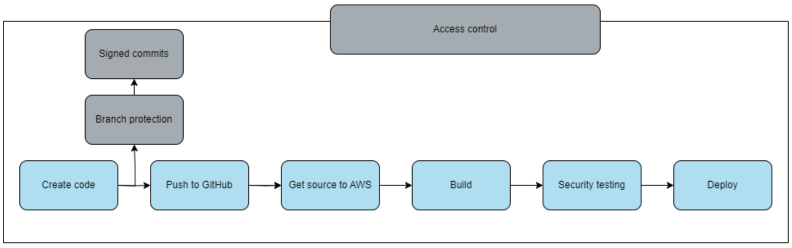


Figure 3.7: Pipeline with implemented access control

3.4 Security in maintenance

Once the deployment process is complete, the application is transferred to the cloud environment in Amazon Web Services [52]. It is essential at this stage to keep the security up to date to ensure that the application and data are protected. AWS offers a range of best practices organizations can follow to decrease risks associated with cloud computing and ensure that the AWS environment is secure.

After successfully completing the deployment process, it is important to ensure infrastructure security and performance. To solve security problems and offer new features to the system, regular maintenance and upgrades are required. Furthermore, frequent backups and disaster recovery testing are required to protect data and applications. Maintaining and testing the system on a regular basis helps prevent downtime and improve system dependability.

It is critical to monitor cloud-based applications continuously in order to discover issues or potential risks This includes identifying errors, performance problems, safety weaknesses, and other relevant issues. AWS provides specific monitoring tools, such as AWS Cloudtrail[1] and AWS X-ray[2].

During the development phases, various security tests, such as security scans and penetration testing, should be done to discover possible vulnerabilities and resolve them before release. However, it is just as critical to do post-deployment testing to ensure that any additional vulnerabilities are found and resolved as soon as possible. Regular security scans and penetration testing can lower the danger of exploitation significantly. By running these tests regularly, a company can keep track of any potential security issues and take preventive measures to mitigate them.

To ensure that the application is protected, it is recommended that the organization implement a web application firewall like AWS WAF[3], to prevent malicious application attacks such as SQL-injection, cross-site scripting and other attacks. AWS's WAF service offers a managed set of protective rules, allowing customized rules and access control lists based on the company's needs and risk models. This makes it possible to provide web application security with more customization and specification. Additionally, it is important to take steps to prevent DDoS attack which can be done using AWS Shield[4].

Maintaining security is crucial during the maintenance phase of the SDLC, which starts after the application has been deployed. It is critical to keep up with regular maintenance and testing during this phase and take proactive measures to mitigate any issues that may arise. Organizations can identify and address potential security vulnerabilities beforehand to prevent them from becoming significant issues. This is particularly crucial when deploying applications to the cloud due to the complex and ever-changing security environment.

[1] Available at: https://aws.amazon.com/cloudtrail/
[2] Available at: https://aws.amazon.com/xray/
[3] Available at: https://aws.amazon.com/waf/
[4] Available at: https://aws.amazon.com/shield/

Best practices such as regular security audits, vulnerability scans, and patches can ensure the application remains secure and protected against potential threats. Monitoring for unusual or suspicious activity can also aid in detecting and preventing security breaches. Organizations can help ensure their AWS applications' continued reliability and security by prioritizing security throughout the maintenance phase.

3.5 Finished pipeline

Figure 3.8 illustrates a finished pipeline after all security measures are included. This also includes maintenance.

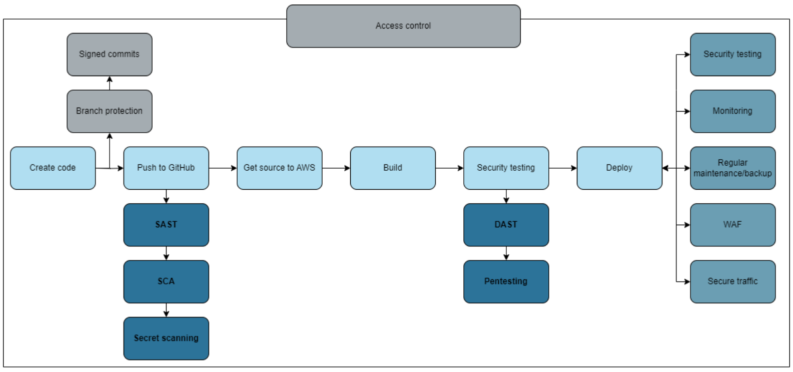


Figure 3.8: Pipeline with all security measures implemented

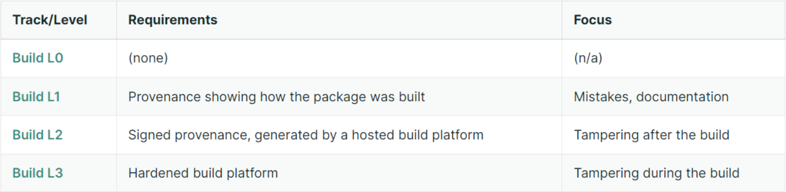
3.6 Frameworks

3.6.1 Introduction

A framework is *"a supporting structure around which something can be built"* [53], and can be used in connection with securing the SDLC. Frameworks outlined in this chapter are not intended to be rigid regulations but rather valuable recommendations for software developers to enhance security measures. By implementing these suggestions, developers can confidently improve the security of their software.

3.6.2 Supply-chain Levels for Software Artifacts

Supply-chain Levels for Software Artifacts (SLSA)[5] is a framework for securing the software supply chain created by Google in collaboration with OpenSSF[6] [54]. The framework is made into a common vocabulary checklist for developers to evaluate the security of the software they are creating. SLSA is organized into tracks and levels. The levels refer to the increasing security guarantee of the supply chain, the highest level being level 3. The levels are split further into tracks. Tracks are certain aspects of the supply chain, for example, the Build track, which currently is SLSAs only track.



Track/Level	Requirements	Focus
Build L0	(none)	(n/a)
Build L1	Provenance showing how the package was built	Mistakes, documentation
Build L2	Signed provenance, generated by a hosted build platform	Tampering after the build
Build L3	Hardened build platform	Tampering during the build

Figure 3.9: SLSA levels for the Build track
[55]

Currently, there are three levels split into one track. To achieve the different build levels, the developers have to do the following:

[5] Available at: https://slsa.dev/

[6] Available at: https://openssf.org/

To achieve SLSA Build Level 1, the developers must use a consistent build process, which can quickly be adopted. Additionally, it is essential to generate provenance automatically on the build platform, which describes how the artifact was built. This includes information on the entity responsible for building the package, the specific build process used, and the top-level inputs utilized during the build process.

In order to reach SLSA Level 2, all Level 1 requirements must be in place. Further, the build has to be run on a platform that signs the provenance. Finally, this provenance's authenticity must also be verified.

Similarly to Level 2, all previous level requirements must be achieved to get to SLSA Level 3. In addition, the build platform needs to be able to secure the secrets used for signing provenance and prevent any interference between runs from the same project.

3.6.3 Secure Software Development Framework

Secure software development framework (SSDF)[7] is a framework consisting of practices for a secure software development, created by National Institute of Standards and Technology (NIST) [56]. The organization should integrate the SSDF into their already existing software development practices. SSDF does not specify how each practice should be implemented. It emphasizes the outcome of the practices rather than how to perform them. Organizations in any sector or community can use the SSDF, regardless of their size or level of cyber security competence. This framework is intended to be user-friendly and adaptable, making it appropriate for a wide range of businesses with varied levels of cyber security knowledge. Organizations can use the SSDF to adopt secure software development practices and reduce the risk of potential security vulnerabilities. The framework does not introduce new practices or define new terminology. However, it presents a set of high-level practices based on known standards, guidelines, and documents relevant to secure software development practices.

[7] Available at: https://csrc.nist.gov/Projects/ssdf

The benefits of describing the practices at a high level include that they can be used by organizations in every industry and community, despite their size or level of cyber security knowledge. It can also help companies that buy and use software understand the secure software development methods used by their suppliers. All the practices are described in the framework[8].

There are four groups into which the practices are divided:

- **Prepare the Organization (PO)**: *"Organizations should ensure that their people, processes, and technology are prepared to perform secure software development at the organization level. Many organizations will find some PO practices to also apply to subsets of their software development, like individual development groups or projects."*[56]
- **Protect the Software (PS)**: *"Organizations should protect all components of their software from tampering and unauthorized access."*[56]
- **Produce Well-Secured Software (PW)**: *"Organizations should produce well-secured software with minimal security vulnerabilities in its releases."*[56]
- **Respond to Vulnerabilities (RV)**: *"Organizations should identify residual vulnerabilities in their software releases and respond appropriately to address those vulnerabilities and prevent similar ones from occurring in the future."*[56]

Each practice definition has the following components:

- **Practice**: Name of the practice with a unique identifier, with a description.
- **Task**: One or more steps may be required to carry out a procedure.
- **Notional Implementation Examples**: A selection of tools, procedures, or approaches is presented that may aid in the execution of tasks. It should be noted that these examples are not exhaustive, and their use is not obligatory. In addition, some examples may not be relevant to specific companies or circumstances.
- **References**: References towards established secure development practice documentation and their mappings to specific tasks. Not all references will be relevant in all cases of software development.

[8] Latest version: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf

Chapter 4

Analysis of security tools for the pipeline

4.1 Introduction

The following chapter presents tools that can be utilized as the security tools implemented in the pipeline presented in Chapter 3. Additionally, advantages and disadvantages are presented.

4.2 GitHub security Tools

Below are the GitHub tools that can be utilized to conduct a wide range of application security testing on the code sent through the pipeline.

4.2.1 CodeQL

GitHub has an in-built code scanning tool called CodeQL that allows the users to analyze the code in the GitHub repository to find vulnerabilities and errors in the code [57]. This tool can be used as the SAST tool in the pipeline. Microsoft's best practice for secure SDLC, which argues using approved tools, has listed CodeQL as a recommended tool [39]. The results of these analyzes are shown as code-scanning alerts in GitHub. This feature helps identify existing issues and prevents new ones from being introduced.

CodeQL can be scheduled to run on chosen days or occurrences of events. For example, rather than scanning each branch individually, it is possible to set up a trigger that will initiate the code scan only when the code is pushed to the main branch or when a pull request is made. Limiting the triggers helps to reduce the amount of time and resources required to perform the scans. Also, it minimizes the risk of vulnerabilities or errors being introduced into the production environment.

Any issues found during the scanning process are displayed as alerts within the repository, allowing developers to divide the different issues easily between team members [58]. Once a user fixes the code that triggered the alert, it is automatically closed. Additionally, users can monitor the results of code scanning across their repositories or organization using webhooks and the code scanning API.

4.2.1.1 CodeQL: advantages and disadvantages

These advantages and disadvantages combine the personal experiences and information found during research.

Advantages	Disadvantages
- **Customize triggers:** Teams can decide when to trigger the scanning. This is usually when an event occurs, such as pull requests - **Suitable for projects:** The scan can be configured so that it tailors the different needs - **Auto-build:** When code scanning runs, it automatically uploads the vulnerabilities it found to the repository's security tab	- **Precision:** The precision can depend on the code. If the code requires a specific type of customization, then the precision of the default setup for CodeQL will not necessarily be suitable - **Language support:** Only supports a smaller set of languages

Table 4.1: Advantages and disadvantages of CodeQL

4.2.2 Dependabot

Dependabot is an in-built GitHub tool that helps developers keep their project dependencies up-to-date and is an example of an SCA tool that can be used in the implementation phase [59].

Dependencies can be updated over time as new versions are released. Therefore, developers must keep the dependencies up-to-date to ensure the project stays secure. However, keeping track of all updates that come and manually running these updates can be time-consuming and error-prone. Dependabot automates checking for new versions of the dependencies used in the code and then creates a pull request to update them. The user can then review the updates and see if it is necessary to make any changes. Dependabot can also automatically resolve any conflict that may arise when updating dependencies.

Dependabot uses the "GitHub Advisory Database" to check for vulnerable data [60]. This database covers a lot of public vulnerabilities, and it uses multiple sources, like Common Vulnerabilities and Exposures, explained in 2.7.2, National Vulnerability Database, and several others.

4.2.2.1 Dependabot: advantages and disadvantages

These advantages and disadvantages combine the personal experiences and information found during research [61].

Advantages	**Disadvantages**
- **Automated:** It automates dependency updates, saving time and reducing manual errors - **Support:** It supports a wide range of languages and package managers - **Change-logs:** It provides detailed change-logs and release notes	- **Merge conflict:** Can create merge conflicts with other changes - **False-positives:** May generate many false positives, which can be time-consuming to rectify. Getting false positives requires users to manually review each issue to determine whether a change is necessary.

Table 4.2: Advantages and disadvantages of Dependabot

4.2.3 Secret Scanner

To prevent fraudulent use of accidentally committed secrets, GitHub's secret scanner scans repositories and archived repositories for known secrets [42]. Secret Scanner is provided in two forms:

- **Secret Scanning alerts for partners**
  When a repository is made public, or changes have been pushed to a public repository on GitHub, the code is automatically scanned for any secrets that match the patterns of GitHub's partners. The Secret Scanner also scans for credentials in the public package registry, like the npm registry[1]. GitHub notifies the associated service provider when the scanner detects a secret. The provider will validate the secret and decide what the course of action will be, for example, revoking the secret or creating an issue. The specific action will depend on the level of risk involved.

- **Secret Scanning alerts for users**
  Secret Scanning alerts for users are free for all public repositories. By enabling secret scanning for a repository, the scanner will look for patterns in the code that could match secrets by different service providers. Once the scan is complete, GitHub sends an email alert to the enterprise and the owners of the organizations. However, if a secret has been compromised, GitHub generates an alert for secret scanning.

[1] Available at: https://www.npmjs.com/

4.2.3.1 Secret Scanner: advantages and disadvantages

These advantages and disadvantages combine the personal experiences and information found during research.

Advantages	Disadvantages
- **Free:** GitHub has given public access to this feature - **Convenience:** As projects become more complex and it becomes harder to keep track of all the secrets stored in the repository, Secret Scanner will help detect these faster - **Secure:** Secrets can be easily missed when writing large amounts of code. Enabling Secret Scanning can potentially increase the security	- **False positives:** Secret Scanning cannot always determine if the secret is legitimate or not, which occasionally can occur in false positives - **Limited Coverage:** It can be limited and does not always detect all secrets created manually, such as personal passwords

Table 4.3: Advantages and disadvantages of GitHub´s Secret Scanner [62]

4.3 OWASP ZAP

OWASP Zed Attack Proxy (OWASP ZAP)[2] is an open-source web application security scanner [63]. It is free and maintained by volunteers worldwide under the Open Worldwide Application Security Project. ZAP is a DAST tool and is designed to test web application security. The tool offers functionality for many people - from developers to experienced testers. It is available in versions that are compatible with major operating systems, as well as Docker[3], which means that users are not limited to a specific operating system when using the tool.

4.3.1 OWASP ZAP: advantages and disadvantages

The following advantages and disadvantages combine the personal experiences and information found during research [64].

Advantages	**Disadvantages**
- **Configuration:** Easy to configure with AWS - **Many approaches available:** There are multiple application security testing approaches available that can help discover potential vulnerabilities	- **Documentation:** The documentation could be improved and is, for some, difficult to understand - **Restricted features:** Compared to other tools, the automated scanning features are restricted

Table 4.4: Advantages and disadvantages of OWASP ZAP

[2] Available at: https://www.zaproxy.org/
[3] Available at: https://www.docker.com/

4.3.2 Branch Protection

Branch protection is a feature of GitHub that enforces different rules and requirements for specific branches in the repository [65]. The purpose of branch protection is to maintain the code's security, which is done by ensuring that all changes done to the branch have gone through the proper steps before being merged into the main branch. Below are the different branch protection features that can be enabled in GitHub.

4.3.2.1 Require a pull request before merging

Administrators of the repository can add rules to the repository which restrict pull requests to have a specific number of people approving the changes before merging to a protected branch. Administrators can allow code owners and users with written permissions to approve.

Under this type of protection, the "Four Eyes Principle" is applied. Since this type of protection requires that at least two people approve the merge, including the person doing the changes. This principle can be considered a controlling mechanism that improves the quality of the outcome, minimize risk errors, and prevents malicious actions by a single individual.

4.3.2.2 Require status checks before merging

Maintaining high code quality is essential when multiple users collaborate within a shared repository. By enabling the "require status checks to pass before merging" feature, repository administrators can establish specific criteria that must be met before code is merged, such as requiring code approval from at least one team member.

4.3.2.3 Require conversation resolution before merging

When working together on the same repository, it is essential to have clear communication and collaboration. A way to secure this is to enable "require conversation resolution before merging". Enabling this rule allows all discussions regarding, for example, issues or pull requests that need to be properly resolved before any merging happens.

4.3.2.4 Require signed commits

Enabling require "signed commits" can be considered a security measure that ensures that changes in the code have not been tampered with. To be able to have secured signed commit, all commits pushed to the repository must be signed with a GNU Privacy Guard (GPG) key or an Secure Shell (SSH) key.

4.3.2.5 Require deployments to succeed before merging

"Require deployments to succeed before merging" enables users to enforce the passing of various required checks, such as pre-merge checks or automated tests, before allowing a pull request to be merged into the main branch.

4.3.2.6 Lock branch

Lock branch allows users to lock a branch in a repository, preventing changes from being made to the branch. This rule can be helpful if there are situations where the branch needs to be protected from unauthorized changes or to be deleted.

4.3.2.7 Do not allow bypassing the above settings

This feature stops users from bypassing required checks and restrictions in a repository. For example, if an administrator enables a rule that all pull requests must pass reviews and checks before merging, the feature prompts users to comply before making changes to the branch.

4.3.2.8 Restrict who can push to matching branches

This branch protection can be enabled for public repositories owned by a GitHub Free organization and organizations using GitHub Team or GitHub Enterprise Cloud. When enabled, only specific users or teams with specific permissions are allowed to push any changes to the protected branch.

4.3.3 Branch protection: advantages and disadvantages

These advantages and disadvantages combine the personal experiences and information found during research.

Advantages	Disadvantages
- **Code changes:** It prevents unauthorized code changes, ensuring only authorized users can make changes to a particular branch - **Code quality:** It can help enforce code review and approval of code, specifying the code's quality - **Collaboration:** By restricting certain actions within different branches, users can work on different aspects of the project without compromising others' work	- **Create bottlenecks:** If only a few individuals are authorized to make changes, it can create bottlenecks and delays in approving different issues - **Workflow:** Branch protections can quickly ruin workflows if something always needs approval before moving on

Table 4.5: Advantages and disadvantages of GitHub's Branch Protection

4.4 Amazon Web Services Tools

The following are the AWS tools that can be utilized in creating and deploying a web application. A range of different tools are available, but only the selected tools listed below will be utilized in the deployment process detailed in Chapter 5.

4.4.1 AWS CodePipeline

AWS CodePipeline is a "*...fully managed continuous delivery service that helps you automate your release pipeline. It allows users to build, test, and deploy code into a test production environment...*" [66].

CodePipeline automates the entire pipeline, including the build, test, and deploy phases, and triggers these processes whenever changes are detected in the repository. When a developer pushes changes to the repository, CodePipeline automatically detects them and initiates the process by building them. If any tests are configured, CodePipeline also runs these tests [67].

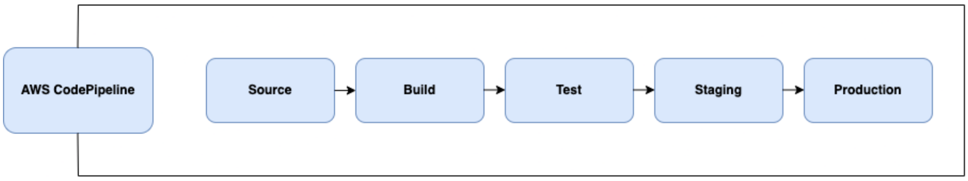


Figure 4.1: AWS pipeline process
Adapted from: [68]

4.4.2 AWS CodeBuild

AWS CodeBuild is described as a "*...fully managed continuous integration service that compiles source code, runs tests, and produces ready-to-deploy software packages.*" [69].

AWS CodeBuild downloads the source code provided to it into a build environment and then uses a buildspec, which defines how the built project should be executed [70].

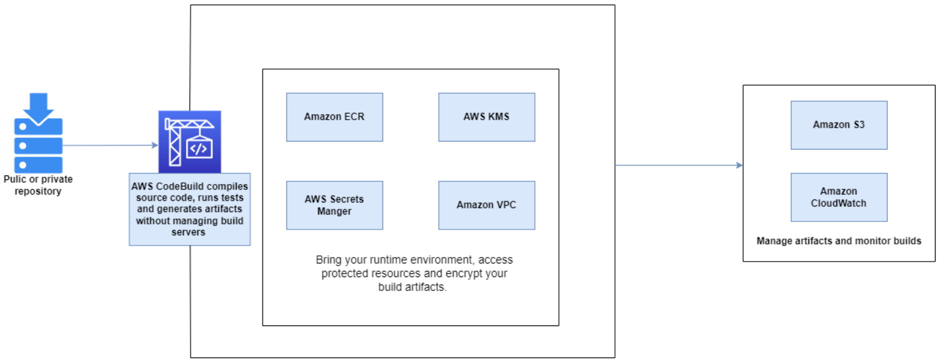

Pulic or private repository
AWS CodeBuild compiles source code, runs tests and generates artifacts without managing build servers
Amazon ECR
AWS KMS
AWS Secrets Manger
Amazon VPC
Bring your runtime environment, access protected resources and encrypt your build artifacts.
Amazon S3
Amazon CloudWatch
Manage artifacts and monitor builds

Figure 4.2: AWS CodeBuild process
Adapted from: [69]

4.4.3 AWS CodeDeploy

AWS CodeDeploy is explained as "*...a fully managed deployment service that automates software deployments to various compute services, such as Amazon Elastic Compute Cloud (EC2), AWS Lambda and more...*" [71]. AWS CodeDeploy helps developers avoid downtime during deployment. It also handles the updating phase of the applications.

CodeDeploy can deploy code that runs on a server and is stored in, for example, GitHub repositories or in an AWS S3 Bucket. In order to use CodeDeploy, developers are not required to make any adjustments to their existing code [72].

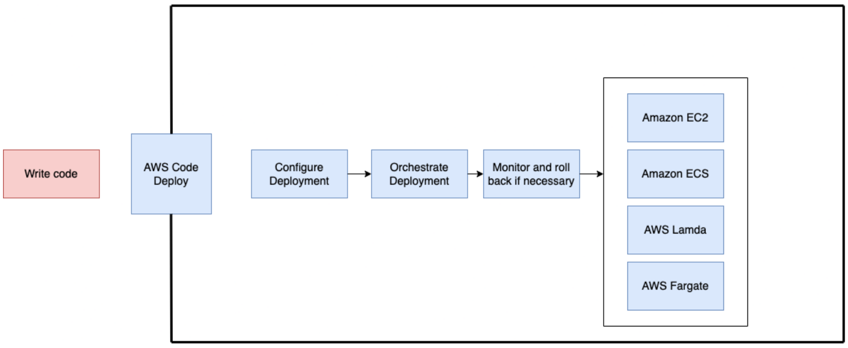


Figure 4.3: AWS CodeDeploy process
Adapted from: [72]

4.4.4 Amazon S3 buckets

Amazon S3 buckets are simple, cloud-based storage resources [73]. S3 buckets are designed to provide users with scalable, durable, and highly available storage, which can be used to store different types of data. Such data can be documents, artifacts, and source code.

4.4.5 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) offers a compute platform, with virtual computing environments, also known as instances [74]. Amazon EC2 offers various instance types to meet computing needs. The instance type determines the instances' CPU, memory, storage, and networking capacity. The instances are launched from Amazon Machine Images (AMIs). AMIs are templates that contain the necessary software configurations and operating system to run the server [75].

Chapter 5

Building a secure end-to-end pipeline

5.1 Introduction

This chapter has a walk-through on one of many ways to set up a secure pipeline. The pipeline starts with a source code, which will be explained further, and pushes the code to GitHub and further to AWS. Within these steps are the security measures and scans discussed previously. Some of these steps are automated using Terraform[1], which is an infrastructure as code tool using HashiCorp Configuration Language as the programming language [76]. Terraform AWS Documentation[2] has been used to set up the different steps.

Figure 5.1 illustrates the pipeline in AWS, showing its structure and connection. The model is built as the different steps in the pipeline are presented.

[1] Available at: https://www.terraform.io/

[2] Available at: https://registry.terraform.io/providers/hashicorp/aws/latest

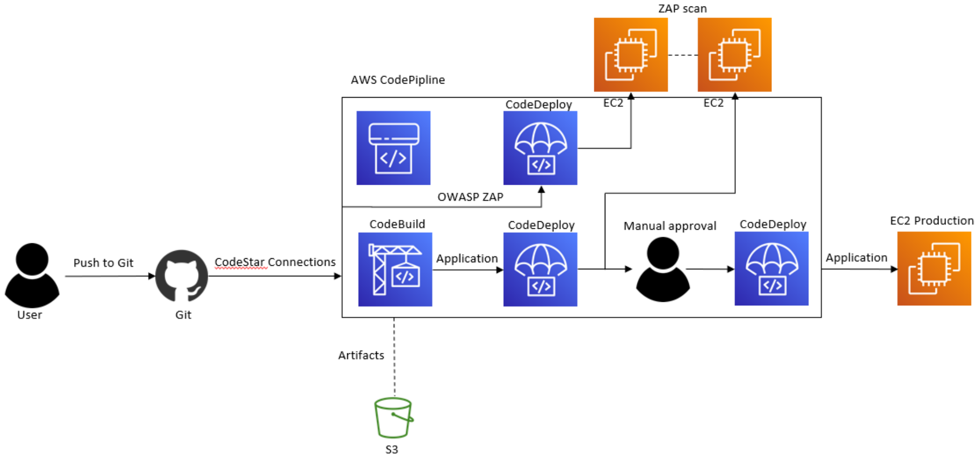


Figure 5.1: Pipeline in AWS

5.2 Code used in the pipeline

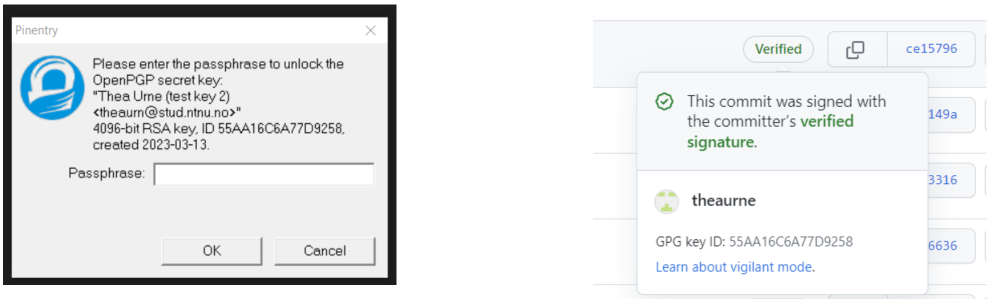
For testing, the group decided on using OWASP Juice Shop[3], which is a deliberately vulnerable web application that is designed to help developers and others learn about web application security concepts. The code is designed to simulate a real-world application by having common vulnerabilities within the code. The intention is to encourage users to find and exploit these vulnerabilities and increase users' understanding of web application security [77]. The code in the OWASP Juice Shop is open source code on GitHub and is written in TypeScript, which uses a Node.js server and Angular. Angular is a TypeScript-based application framework for Front-end [78]. The code contains different vulnerabilities, including SQL-injections, and cross-site scripting. According to its website, it has registered 101 vulnerabilities in the code.

[3] Code available at: https://github.com/juice-shop/juice-shop

5.3 Pushing to GitHub

Security measures must be in place when the source code is ready to be pushed to GitHub. One of the previously mentioned measurements is branch protection. Branch protection rules are easy to enable in GitHub. Enabling the rules are done for each repository. GitHub has different branch protection rules that can be enabled, which are explained in Section 4.3.2. Enabling the rules can be done in GUI or automated using Terraform, as shown in in the GitHub repository [4].

Further, the commit signature has to be configured. For signing commits in GitHub, either SSH or GPG keys can be used. The keys need to be generated and connected to the user's GitHub account. GitHub provides documentation[5] on how to create GPG and SSH keys, as well as how to connect these keys to a GitHub account.



Pinentry

Please enter the passphrase to unlock the OpenPGP secret key:
"Thea Urne (test key 2)
<theaurn@stud.ntnu.no>"
4096-bit RSA key, ID 55AA16C6A77D9258,
created 2023-03-13.

Passphrase:

This commit was signed with the committer's verified signature.

theaurne

GPG key ID: 55AA16C6A77D9258

Learn about vigilant mode.

(a) Required passphrase when committing to GitHub

(b) Verified commit

Figure 5.2: Required signed commits

Figure 5.3: Pushing the source code to Git

[4] Code for enabling branch protection rules available at: https://github.com/DCSG2900-Bachelor-thesis/Enable_Branch_Protection.git

[5] Instructions on making GPG or SSH keys available at https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits

5.4 Managing security in GitHub

When the source code is securely pushed to its belonging branch and is being merged into the main branch, CodeQL needs to scan the code for vulnerabilities. This is done by setting up CodeQL[6] and customizing the trigger for the code scan.

Setting up CodeQL is done by adding the CodeQL YAML file to the workflow, which contains the configuration settings for running the analysis. Adding the pull request and push events to the workflow file customizes the trigger for the analysis, causing the analysis to run automatically whenever a pull request or push is made to the specified branch [79]. The branch specified in Code 5.1 is the main branch.

```
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
```

Code 5.1: Custom trigger for CodeQL alerts

An example on how to enable CodeQL from the CLI is shown in the GitHub repository [7].

Other security tools in GitHub are Dependabot[8] and Secret Scanner[9]. These are also enabled in the security settings. Alternatively, Dependabot can be enabled from the CLI [10]. After being enabled, all these tools automatically scan the code for vulnerabilities and secrets.

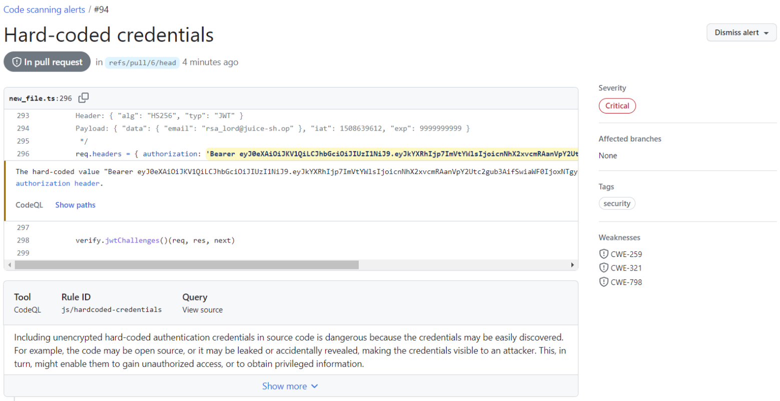
[6] Instructions on enabling CodeQL available at https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/configuring-code-scanning-for-a-repository

[7] Instructions on how to enable CodeQL from cli: https://github.com/DCSG2900-Bachelor-thesis/Enable_CodeQL

[8] Instructions on how to enable Dependabot available at: https://docs.github.com/en/code-security/dependabot/dependabot-security-updates/configuring-dependabot-security-updates

[9] Instructions on how to enable Secret Scanner available at: https://docs.github.com/en/code-security/secret-scanning/configuring-secret-scanning-for-your-repositories

[10] Instructions on how to enable dependabot from cli: https://github.com/DCSG2900-Bachelor-thesis/Enable_dependabot



Code scanning alerts / #94

Hard-coded credentials

In pull request in `refs/pull/6/head` 4 minutes ago

`new_file.ts:296`

```
        Header: { "alg": "HS256", "typ": "JWT" }
        Payload: { "data": { "email": "rsa_lord@juice-sh.op" }, "iat": 1508639612, "exp": 9999999999 }
         */
        req.headers = { authorization: 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjp7ImVtYWlsIjoicnNhX2xvcmRAanVpY2Ut
```

The hard-coded value "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjp7ImVtYWlsIjoicnNhX2xvcmRAanVpY2Utc2gub3AifSwiaWF0IjoxNTgy authorization header.

CodeQL Show paths

```

        verify.jwtChallenges()(req, res, next)

```

Tool	Rule ID	Query
CodeQL	js/hardcoded-credentials	View source

Including unencrypted hard-coded authentication credentials in source code is dangerous because the credentials may be easily discovered. For example, the code may be open source, or it may be leaked or accidentally revealed, making the credentials visible to an attacker. This, in turn, might enable them to gain unauthorized access, or to obtain privileged information.

Severity
Critical

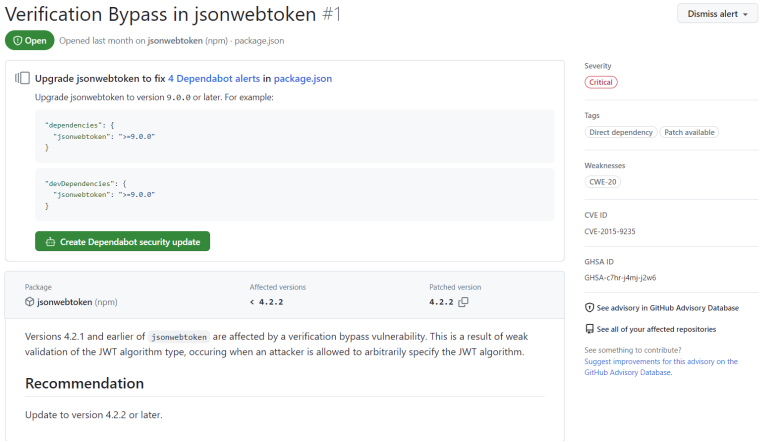
Affected branches
None

Tags
security

Weaknesses
CWE-259
CWE-321
CWE-798

Figure 5.4: Critical alert from CodeQL

An example of code scanning alerts using CodeQL is the critical alert shown in Figure 5.4, notifying the user of a hard-coded credential. The alert is shown in a pull request from a branch to the main branch. The code scanning alerts refer to CWE weaknesses, which are explained further in Section 2.7.3.



Verification Bypass in jsonwebtoken #1

Open Opened last month on **jsonwebtoken** (npm) · package.json

Upgrade jsonwebtoken to fix 4 Dependabot alerts in package.json

Upgrade jsonwebtoken to version `9.0.0` or later. For example:

```
"dependencies": {
  "jsonwebtoken": ">=9.0.0"
}
```

```
"devDependencies": {
  "jsonwebtoken": ">=9.0.0"
}
```

Package	Affected versions	Patched version
jsonwebtoken (npm)	< 4.2.2	4.2.2

Versions 4.2.1 and earlier of `jsonwebtoken` are affected by a verification bypass vulnerability. This is a result of weak validation of the JWT algorithm type, occuring when an attacker is allowed to arbitrarily specify the JWT algorithm.

Recommendation

Update to version 4.2.2 or later.

Severity
Critical

Tags
Direct dependency Patch available

Weaknesses
CWE-20

CVE ID
CVE-2015-9235

GHSA ID
GHSA-c7hr-j4mj-j2w6

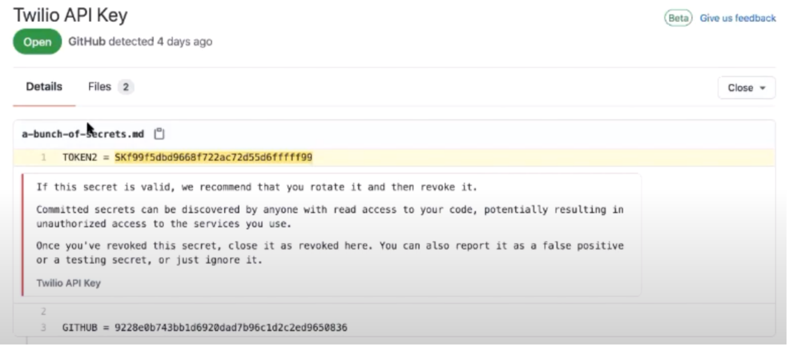
See advisory in GitHub Advisory Database

See all of your affected repositories

See something to contribute?
Suggest improvements for this advisory on the GitHub Advisory Database.

Figure 5.5: Critical alert from Dependabot

Figure 5.5 shows one of the critical alerts from Dependabot in the Juice-Shop repository. It shows a vulnerability in the dependency version used, following a suggestion on a patched version. Dependabot, similarly to CodeQL, refers to CWE weaknesses. Additionally, it refers to a CVE ID, which is explained further in Section 2.7.2.



```
TOKEN2 = SKf99f5dbd9668f722ac72d55d6ffff99
```

If this secret is valid, we recommend that you rotate it and then revoke it.

Committed secrets can be discovered by anyone with read access to your code, potentially resulting in unauthorized access to the services you use.

Once you've revoked this secret, close it as revoked here. You can also report it as a false positive or a testing secret, or just ignore it.

Twilio API Key

```

GITHUB = 9228e0b743bb1d6920dad7b96c1d2c2ed9650836
```

Figure 5.6: Alert from secret scanner
[80]

The GitHub Secret Scanner did not detect any secrets in the Juice-Shop repository. Therefore, Figure 5.6 is taken from GitHub's demonstration of the secret scanner [80]. The alert warns the user about an API key that matches the pattern of their partner Twilio.

5.5 Retrieving the source code in AWS

When the source code is pushed to the main branch in the GitHub repository, the source code can be built in AWS. For this to be done, a connection between the AWS and the GitHub account needs to be made. The connection can be made using AWS CodeStar Connections[11]. The connection is used by AWS CodePipeline to automatically trigger the pipeline in response to any changes made to the GitHub repository. A connection can be created on the AWS website or by using Terraform to create a connection resource with GitHub as the provider.

```
resource"aws_codestarconnections_connection""code-connection"{
  name = "code-connection"
  provider_type = "GitHub"
}
```

Code 5.2: Creation of a connection between AWS and GitHub

[11] A tool that enables CodePipeline to connect to third-party repositories [81].

After successfully creating the connection resource, it is necessary to perform a manual verification process in AWS. This step is crucial because it is impossible to pass secrets through AWS to GitHub login, as the connection opens a third-party login page that AWS does not control.

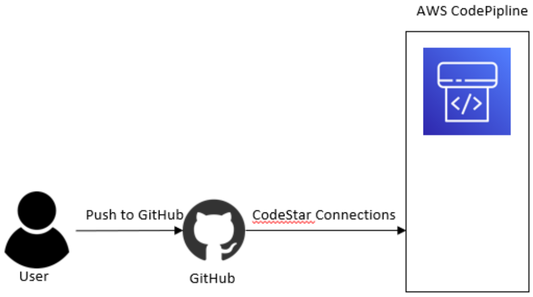


Figure 5.7: Committing the source code to AWS

5.6 Storing artifacts

To store artifacts generated during the different stages of the pipeline, an S3 bucket is created. The bucket is created as a resource with optional variables. One crucial aspect to consider is the need for unique bucket names within the server area, as AWS uses this to differentiate between buckets owned by different users. The name can be configured, like in Code 5.3, or the name variable can be excluded, and AWS will configure the name themselves.

```
resource "aws_s3_bucket" "codepipeline_artifact" {
  bucket = "artifact-bucket-unique-name"
}
```

Code 5.3: Creation of an S3 Bucket

Granting access to the bucket is necessary for each stage in the pipeline to retrieve the previous stage's artifacts from the S3 bucket. Therefore, access rights need to be granted separately for each stage in the pipeline. Code 5.4 grants the CodePipeline access to S3 rules, like access to the bucket itself.

```
statement {
    sid = ""
    actions = [
      "s3:GetObject",
      "s3:PutObject",
      ]
    resources = [
    "${aws_s3_bucket.codepipeline_artifact.arn}/*"
    ]
    effect = "Allow"
}
```

Code 5.4: Permissions to CodePipeline

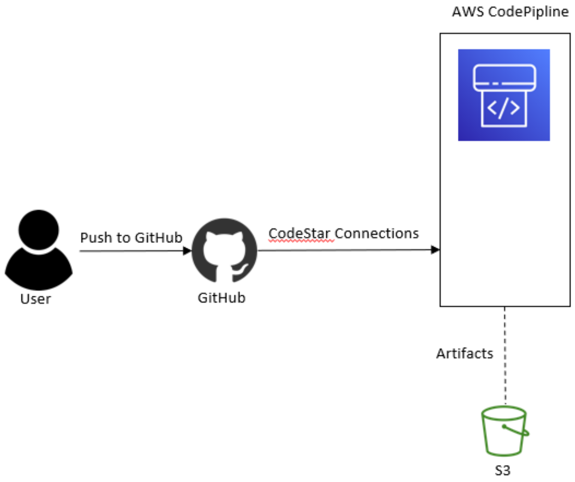


Figure 5.8: Storing the artifacts in an S3 Bucket

5.7 Build stage

In the build stage, CodePipeline uses CodeBuild to compile the source code into a working website [82]. For the CodeBuild to run, the operating system, image, compute type and compute environment must be specified. In this example, the environment type is set to be AWS's own Linux container.

The compute type refers to the amount of memory, CPUs, and disk space needed, which in Code 5.5 is set to be the least amount of hardware resources. The combination of these variables is called a “build environment”.

```
environment {
    compute_type = "BUILD_GENERAL1_SMALL"
    image = "aws/codebuild/standard:6.0"
    type = "LINUX_CONTAINER"
    image_pull_credentials_type = "CODEBUILD"
}
```

Code 5.5: Creation of a build environment

Further, CodeBuild downloads the source code from the S3 bucket, which was added from the previous stage. With the source code, a buildspec file needs to be specified to run the build.

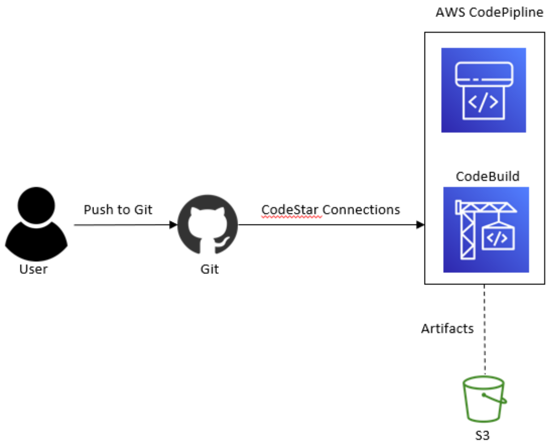


Figure 5.9: CodePipeline using CodeBuild

5.8 Deployment to testing

5.8.1 Setting up CodeDeploy

In order to configure CodeDeploy, the user must create an application and a deployment group. The deployment group contains settings and configurations used during the deployment. In Code 5.6, the settings specify a preset configuration. Such configuration is a set of rules and conditions, like the compute platform and conditions for failure [83]. The deployment group also specifies which preconfigured instance to use for the deployment[12].

```
resource "aws_codedeploy_deployment_group" "deploy_group" {
  deployment_group_name = var.deploy_group_name
  deployment_config_name = aws_codedeploy_deployment_config.
                          deployment_config.id
  app_name = "deployment"
  service_role_arn = aws_iam_role.codedeploy-role.arn

  ec2_tag_filter {
    key = "Name"
    type = "KEY_AND_VALUE"
    value = "test_instance"
  }
}
```

Code 5.6: Creation of the deployment group

[12] Setting up an instance is available at https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

As seen in Code 5.6, the deployment group specifies the application. The application is *“a name or container used by CodeDeploy to ensure the correct revision, deployment configuration, and deployment group are referenced during a deployment”* [84]. Therefore, to configure the application, it is only necessary to provide a name for it, as shown in Code 5.7.

```
resource "aws_codedeploy_app" "codedeploy" {
  name = var.codedeploy_app_name
}
```

Code 5.7: Creation of an application for the deployment

The deployment itself deploys the given revision, which typically consists of the source code for an application. In addition, an appspec file is added to the revision.

5.8.2 Setting up for testing

This stage utilizes two Amazon EC2 instances: one for OWASP ZAP and the other for the application itself. Setting the scan and the application on separate instances makes it easier to manage them individually. This setup also allows for code to be reused when deploying the application into production. However, both instances utilize the same S3 bucket, but the files are separated into two different directories so they do not interfere with each other.

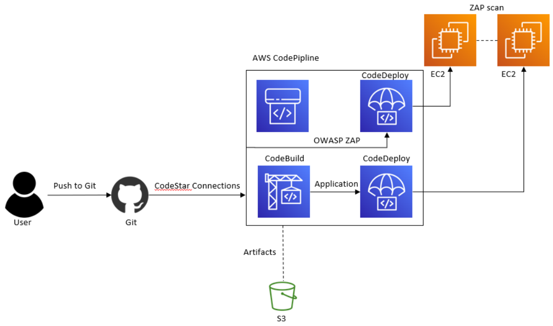


Figure 5.10: Deployment to testing

For OWASP ZAP to be able to scan the application, the instances need to communicate. To enable communication between the Amazon EC2 instances within the Virtual Private Cloud (VPC), individual network interfaces have been allocated to each instance with a single IP address. This method guarantees that both instances can find each other because the IP address will always be the same.

```
resource "aws_network_interface" "interface_network1" {
  subnet_id = aws_subnet.my_subnet.id
  private_ips = ["172.16.10.100"]
}
resource "aws_network_interface" "interface_network2" {
  subnet_id = aws_subnet.my_subnet.id
  private_ips = ["172.16.10.101"]
}
```

Code 5.8: Allocation of IP adresses

The security tests are conducted by setting up a docker container on the Amazon EC2 instance created for OWASP ZAP, which runs one of the automated standard tests predefined by the DAST tool. In addition, the container uses the "weekly stable image" of OWASP ZAP[13]. Although these tests are typically customized to suit an application's specific requirements, the predefined tests are satisfactory for this demonstration. Once the container is launched, the scan is directed towards the other instance in the VPC, and the output obtained is shown in Code 5.9 (full output in Appendix A). In addition OWASP ZAP gives a full human-readable report in HTML format, shown in Appendix A.1.

[13] Available at https://hub.docker.com/r/owasp/zap2docker-stable/

```
WARN-NEW: Timestamp Disclosure - Unix [10096] x 1
       http://172.16.10.100:3000/main.js (200 OK)
WARN-NEW: Cross-Domain Misconfiguration [10098] x 11
       http://172.16.10.100:3000/ (200 OK)
       http://172.16.10.100:3000/assets/public/favicon_js.ico (200 OK)
       http://172.16.10.100:3000/ftp (200 OK)
       http://172.16.10.100:3000/ftp/acquisitions.md (200 OK)
       http://172.16.10.100:3000/main.js (200 OK)
WARN-NEW: Modern Web Application [10109] x 13
       http://172.16.10.100:3000/ (200 OK)
       http://172.16.10.100:3000/home/build/routes/fileServer.js:16:13
           ↪  (200 OK)
       http://172.16.10.100:3000/home/build/routes/fileServer.js:32:18
           ↪  (200 OK)
       http://172.16.10.100:3000/home/build/routes/runtime.js (200 OK)
       http://172.16.10.100:3000/home/node_modules/express/lib/router/
           ↪ index.js:280:10 (200 OK)
WARN-NEW: Dangerous JS Functions [10110] x 2
       http://172.16.10.100:3000/main.js (200 OK)
       http://172.16.10.100:3000/vendor.js (200 OK)
FAIL-NEW: 0 FAIL-INPROG: 0 WARN-NEW: 9 WARN-INPROG: 0 INFO: 0 IGNORE:
    ↪ 0 PASS: 51
```

Code 5.9: The output of an OWASP ZAP baseline scan

5.9 Deployment to production

In the production stage, CodePipeline utilizes CodeDeploy, as described in Section 5.8. Before the code gets moved to production, it must be accepted by manual approval by a user in AWS. The purpose of this step is to allow for a review of the results from the automated security tests, to ensure that no faulty code is deployed. When new code or changes have gone through testing, a notification will occur in the Amazon SNS topic[14] that is created. Someone then has to approve the code waiting to be moved to production.

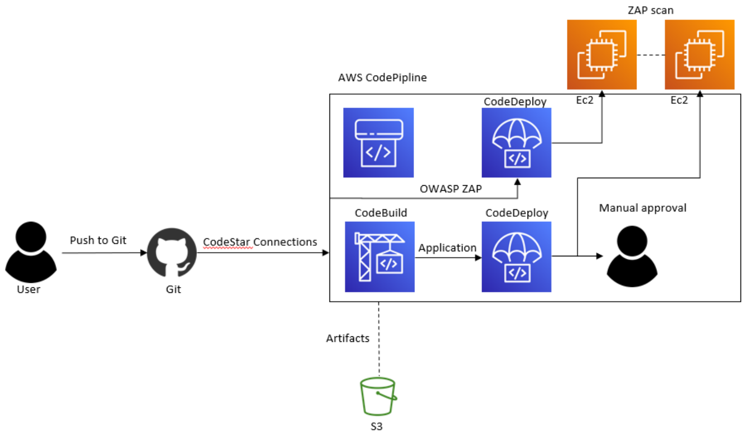


Figure 5.11: Manual approval before deploying the application

[14] Amazon Simple Notification Service (Amazon SNS) provides message delivery to a topic. A topic is the communication channel the users can subscribe to [85].

After the manual approval process is completed, the code will be deployed to production in a similar manner to the process described in Section 5.8. Figure 5.12 illustrates the flow of the pipeline after completing all the necessary steps.[15]

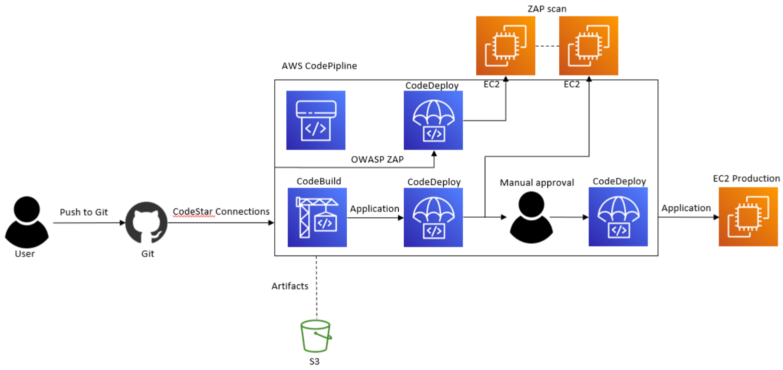


Figure 5.12: Finished pipeline created in AWS

[15] Complete code available at: https://github.com/DCSG2900-Bachelor-thesis/CodePipeline

Chapter 6

Discussion

6.1 Introduction

This chapter details the reasons behind the group's decision-making process during the pipeline construction. It also covers any modifications made to the project and highlights any limitations encountered.

6.2 Implementation of the SDLC

Throughout the thesis, the group created a report and code, and while doing so, they chose to explore the implementation of the SDLC approach.

During the planning phase, the group focused on gathering requirements for the thesis and developing a project plan. Initially, the group created a plan for using third-party tools for the scans and set an estimate of the costs. However, the plan of using third-party tools was later discarded, except for the DAST tool. The project plan included a Gantt chart outlining the expected task timeframe. Additionally, the group utilized Jira[1] to create a Kanban board for task assignments. The group created several models to prepare for pipeline implementation that detailed its structure, including security scans. The group also became familiar with the software and tools that would be utilized for the pipeline.

[1] Available at https://www.atlassian.com/software/jira

In the implementation phase, the group began the development of the pipeline while simultaneously working on the thesis. The coding process for the pipeline involved repeatedly implementing code and testing its functionality. For instance, the group implemented the code for AWS CodeBuild and tested whether the service was created in accordance with the code. As a result, the implementation and testing phases were carried out iteratively.

The result was ultimately merged into a GitHub repository and documented in the thesis. However, as the product is not an application and is not intended for direct use in the future but rather as a demonstration, plans have yet to be made for its maintenance.

6.3 Chosen branch protection rules

When deciding which branch protection rules to implement, it is essential that securing the branch is relatively manageable for the workflow. Several possible protection rules to enable are described in section 4.3.2. Enabling all rules at once may do more harm than good, every pull request and commit must go through several steps, making the workflow less efficient. Therefore, selecting only the most valuable rules will benefit the development.

Enabling “Require a pull request before merging” is based on the “Four Eyes Principle”, which is described more in detail in section 3.3 [86]. By enabling this rule, the risk of unwanted code being merged into the basecode is lower, without disrupting the work of all developers since only two need to look through the code.

Another implemented branch rule requires signed commits. Signed commits require some pre-configurations, but once that is done, all the developers must enter a passphrase, and the commit is verified. Implementing required signed commits ensures the integrity of the code while the workflow is undisturbed.

6.4 The different tools chosen

After evaluating multiple tools for securing the pipeline, the group has given recommendations regarding the various scans. Although the group mainly decided to use integrated tools within GitHub, the decision was primarily based on their suitability for the given task and Erik Hjelmås' recommendation (see the meeting in Appendix F.1). It is important to note that this does not necessarily imply that others should adopt the same tools. Conducting such tests is essential in application development, and selecting tools for carrying out these tests should be based on specific requirements. Therefore, it is necessary to explore the various tools available for different tests and assess whether they meet one's particular needs. The tools selected by the group are examples of what can be used.

6.4.1 Why OWASP ZAP was chosen

In the course of researching security tools, an evaluation was made of the three tool types: SAST, DAST, and SCA. As neither GitHub nor AWS provides a DAST tool, a third-party tool had to be used. Among these, the OWASP ZAP appeared as the most widely used web application scanner globally [87]. Also, AWS referenced the DAST tool in an example pipeline using open source tools [38]. This tool is convenient due to its user-friendly interface and straightforward setup process, suitable for individuals of all skill levels. Furthermore, it is open-source and available to all users without additional costs. Its integration into AWS pipelines was found to be a simple and uncomplicated process.

6.4.2 Why tools integrated into GitHub were chosen

Initially, the thesis aimed to investigate third-party tools, like Snyk and Mend, which could be integrated into GitHub and run scans on the code pushed to the repository. Then, however, it was decided to look into integrated tools in GitHub. The goal was to explore the potential benefits of using pre-integrated tools to create a secure pipeline and give priority to overall functionality rather than choosing individual tools.

6.4.3 The group's experience with CodeQL

As mentioned in section 4.2.1, the group used CodeQL as their SAST tool. CodeQL is set as default when enabling code scanning, simplifying the setup process by requiring fewer steps than other code scanning tools. Furthermore, using the tool in this project was uncomplicated since the only customisation was the trigger, explained in Section 5.4. However, the group did not explore any further customisation and is therefore not familiar with the level of complication this brings.

CodeQL, in collaboration with Dependabot, found all 101 vulnerabilities reported in the vulnerable-by-design web application OWASP Juice Shop. However, Juice Shop is a widely used repository, which might be why CodeQL and Dependabot managed to find all the vulnerabilities. The group could have considered testing additional repositories for known vulnerabilities to conduct a more comprehensive evaluation of the tool. However, due to the scope limitations, the group decided not to conduct further research on this topic.

Overall, the group found the tool intuitive and appreciated its practicality in providing detailed explanations and concrete examples of patches when detecting errors.

6.4.4 The group's experience with Dependabot

The group utilized Dependabot as an SCA tool, which was relatively straightforward to configure. Similarly to both Dynamic Application Security Testing and Static Application Security Testing, the group conducted an SCA scan on the Juice Shop repository.

Dependabot excels in providing comprehensive explanations of vulnerabilities and suggesting recommendations for resolving them. For instance, if an outdated dependency version is detected, Dependabot advises on the appropriate version to update to. Furthermore, in cases where an update is unavailable, Dependabot suggests that an alternative dependency should be utilized. However, it does not offer specific alternative dependencies for replacement. Nonetheless, the group found this feature beneficial as it provides guidance for maintaining security.

6.4.5 The group's experience with Secret Scanning

The group also looked into Secret Scanning. While Secret Scanning does not fall strictly under the category of application security testing, the group recognized its value in scanning code for sensitive information such as tokens and API keys.

Considering that SAST tools primarily focus on scanning code for vulnerabilities that attackers can exploit, the group found it valuable to utilize an additional scanner that specifically checks for secrets.

Similar to Dependabot, configuring Secret Scanning was also a simple process, requiring only a few steps to enable. Due to the ease of use and the value it brings, explained in Section 3.2.3, the group considered the tool as highly beneficial.

6.4.6 The group's experience with OWASP ZAP

The group selected OWASP ZAP as their DAST tool and found it straightforward to set up. However, instead of customizing their scans, the group relied on pre-made functions due to the lack of extensive documentation on scan customization. Creating customized scans would have been complex and time-consuming, leading to the decision to utilize the pre-existing functions.

Once OWASP ZAP was successfully installed and configured with basic settings, the group faced challenges in comprehending the output results and extracting the output from the docker container.

6.5 Automation

Automation is using infrastructure as code to perform tasks instead of doing them manually [88]. Implementing automation in the organization's systems can eliminate the need for many developers to manage all the different infrastructure elements the organization may have. Furthermore, there are numerous advantages to replacing manual work with automation. It can reduce costs by replacing the manual work of IT professionals with automated processes.

Automation can also speed up development by automating repetitive tasks like testing, building, and deploying code, allowing developers to work on more complex jobs [89]. In addition, it can improve security by reducing the risk of human errors or by automating security testing that can help speed up the detection, verification, and escalation of security issues without requiring manual involvement. The group's decision to use Terraform and construct a pipeline with infrastructure as code is mostly based on the abovementioned benefits. Furthermore, the group attempted to automate as many processes as possible to remove the need for manual involvement during deployment and testing.

Despite the numerous benefits of automating the development processes, the implementation can be time-consuming and require a complex setup. However, once completed, the setup would be significantly less complicated and require less time. One significant benefit of automating the development process is the ability to reuse the automated configuration across several projects [90]. Automating the process once may be reused in various settings, allowing numerous projects to benefit from the same improved pipeline.

A goal should be to create an idempotent process that assures the pipeline's output is consistent every time it is built [91]. Adopting idempotence as a practice in DevOps is an approach during application development that ensures a high-quality experience for both users and software teams. Idempotence eliminates the requirement for post-deployment cleanup, lowering the likelihood of errors.

6.6 The Use of Security Framework

This section concerns incorporating the two frameworks mentioned in Section 3.6 into the thesis, even though it was not initially considered. By examining the principles and requirements of these frameworks, valuable perspectives and insights can be discovered. In addition, by comparing the work with frameworks, one can identify areas where improvement is possible and any missing components.

6.6.1 SLSA

The group tried to implement the SLSA framework in the practical part of this project to enhance the integrity of the code we developed. This led the group to attempt to work through the different levels and requirements mentioned in the section 3.6.2.

SLSA Level 1, requirement 1: *"Software producer follows a consistent build process so that others can form expectations about what a "correct" build looks like."* [55]

This requirement was achieved by creating instructions in the README.md file describing how to build and execute the code in the GitHub project[2].

SLSA Level 1, requirement 2: *"Build platform automatically generates provenance describing how the artifact was built, including: what entity built the package, what build process they used, and what the top-level input to the build were."* [55]

This requirement was not achieved because the group encountered difficulties when trying to find a suitable solution to generate provenance. A sentiment shared by Kelsey Hightower in his talk at Strange Loop 2022 [92] highlighting the primary issue with the SLSA framework's high complexity. This complexity was also encountered by the group while searching for a solution.

SLSA Level 1, requirement 3: *"Software producer distributes provenance to consumers, preferably using a convention determined by the package ecosystem."* [55]

Due to the identical issue encountered in Requirement 2, the group was unable to accomplish Requirement 3.

To conclude, the group could not find a suitable solution for generating provenance. As a result, they were unable to advance to the next level, since all the requirements from the previous level must be fulfilled.

[2] Available at: https://github.com/DCSG2900-Bachelor-thesis/CodePipline/blob/main/README.md

6.6.2 SSDF

While numerous models, such as waterfall, spiral, and agile, exist for the Software Development Life Cycle, only a few prioritize security, therefore, is it necessary to integrate security into the SDLC models. Utilizing a security-focused framework, such as the SSDF, as a reference can serve as a starting point for enhancing the security of the SDLC.

In SSDF documentation, elaborated in section 3.6.3, includes various practices covering different security aspects that can be implemented into the development process. While the SSDF is a suitable framework for secure software development, the group did not explicitly consider it while developing and designing the pipeline in the thesis. However, the group believes that the practical work in the thesis aligns with many of the practices and principles of the SSDF. The group reviewed some examples of the practices they believed to be suitable and their implementation methods. The group has decided to review the practices category-wise, with each practice being assigned to a specific category.

Prepare the Organization (PO): Implement Supporting Toolchains (PO.3): *"Use automation to reduce human effort and improve the accuracy, reproducibility, usability, and comprehensiveness of security practices throughout the SDLC, as well as provide a way to document and demonstrate the use of these practices. Toolchains and tools may be used at different levels of the organization, such as organization-wide or project-specific, and may address a particular part of the SDLC, like a build pipeline."*[56]

This practice recommends selecting the right tools for the toolchain, following the recommended security practices, and utilizing tools to generate artifacts. In order to accomplish this, the group used various security tools to run different security tests, all of which comply with best practices for supply chain security. One of the key security practices the group has implemented is to use code-based configuration by automating the pipeline with Terraform. The automation of the pipeline not only ensures that the pipeline is efficient and practical but also promotes consistency and reduces the likelihood of errors. In AWS, the implementation of this practice is shown by the fact that each stage of the pipeline generates an artifact stored in an S3 bucket.

Protect Software (PS): Protect All Forms of Code from Unauthorized Access and Tampering (PS.1): *"Help prevent unauthorized changes to code, both inadvertent and intentional, which could circumvent or negate the intended security characteristics of the software. For code that is not intended to be publicly accessible, this helps prevent theft of the software and may make it more difficult or time-consuming for attackers to find vulnerabilities in the software."*[56]

This practice focuses on maintaining the integrity and availability of code, as well as proper storage of all code forms. To keep track of the modifications to the code, the group utilizes GitHub's version control and the artifacts generated in AWS if necessary. The group also uses signed commits to ensure code integrity. To follow the "least privilege" principle, the group decided to give each member access based on their assigned tasks. Therefore, the code repository was accessible in read-only to the two members responsible for writing the report. In comparison, the other two members, responsible for practical work, were granted full access.

Produce Well-Secured Software (PW): Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7): *"Help identify vulnerabilities so that they can be corrected before the software is released to prevent exploitation. Using automated methods lowers the effort and resources needed to detect vulnerabilities. Human-readable code includes source code, scripts, and any other form of code that an organization deems human-readable."*[56]

This practice covers code security and provides guidelines for evaluating the source code used or written. The evaluation can be done through manual code review or automated tools. The goal is to identify and correct vulnerabilities before releasing the software to prevent exploitation. The group has incorporated various automated security scans throughout the pipeline, such as SCA, SAST, and DAST.

Respond to Vulnerabilities (RV): Identify and Confirm Vulnerabilities on an Ongoing Basis (RV.1): *"Help ensure that vulnerabilities are identified more quickly so that they can be remediated more quickly in accordance with risk, reducing the window of opportunity for attackers."*[56]

This practice includes doing frequent vulnerability scans, threat assessments, and penetration testing, as well as monitoring and analyzing system logs and network traffic for indications of possible security problems. Various automated security scans, such as SCA, SAST, and DAST, have been implemented throughout the pipeline. These scans help detect potential vulnerabilities and security concerns in the code. Apart from the scans, the group also reflected on the option of application monitoring. This involves conducting assessments post-deployment to ensure the application remains secure and no new vulnerabilities have emerged.

6.6.3 Usefulness of the framework

Both the SLSA and SSDF frameworks may give helpful recommendations to developers trying to improve the security of their products. However, while SLSA has the potential to encourage good code heritage and increase security in more significant open-source projects with larger user bases, its complexity may not be justified in smaller projects. SSDF, on the other hand, provides a flexible way to apply security measures, making it a helpful guide for deciding which measures to deploy. Furthermore, the SSDF may be used by developers to assess the security level of their project without needing an extra step in the development process. Ultimately, the project's unique demands and scope will determine the decision on which framework to use, or whether to use a framework at all.

In retrospect, as the group reflects on the project, they acknowledge that it would have been beneficial to implement the SSDF from the beginning of the project. The group finds it well-suited for smaller projects with a precise scope, providing clear guidelines and making it easier to follow a structured approach. In addition, incorporating the SSDF framework from the start would have facilitated smoother project execution and ensured that the group had specific guidelines tailored to their needs, allowing for a more efficient and focused development process.

6.7 Revising the thesis angle

When writing the project plan, the group initially planned on writing a thesis based on testing security tools and comparing the results. The main focus would be on the tools and testing as many as possible. The group found it challenging to create a unique thesis due to previous theses with similar topics.

After a meeting with the group's professor Erik Hjelmås (F.1), discussing this issue, the group found an alternative approach for the thesis based on a report from Usenix [93] shown to the group. The new approach involved incorporating more practical work with infrastructure as code and focusing on utilizing tools already integrated into GitHub and AWS, and implementing best practice security measures. This approach of the thesis seemed to be more aimed at professional life and what kind of research was needed today.

6.8 Expectations compared to reality

6.8.1 Practical work

As mentioned in Section 1.5.1, the group needed to learn more about infrastructure as code, especially Terraform. During this process, the group faced more issues than anticipated while building the pipeline with Terraform, AWS and GitHub. The group had some experience with Microsoft Azure from previous courses and thought that AWS was more similar to Azure than it is. Additionally, AWS provides a wide range of services that targets the same or similar issues. The process of selecting these services and integrating them has posed a more significant challenge than initially anticipated.

6.8.2 Research

The group struggled to find proper academic research for their thesis. Therefore, the majority of their resources came from websites or blog posts. To establish the credibility of these sources, the group either confirmed the author's or company's credibility or cross-referenced the information with additional sources to confirm its accuracy.

6.9 Critique of the thesis

6.9.1 Not using frameworks from the beginning

The group did not look deeper into following a framework as the scope contained no specifications. However, when the group discovered different frameworks, the group decided that it was too late to integrate them into the work. Doing so would have required a significant amount of work, and since the group was too far into the thesis, it was decided not to look into it—many requirements for the different frameworks needed to be achieved before the thesis was started. Furthermore, many tasks must be completed if the group were to integrate the framework later in the thesis, which would have added more complexity.

6.9.2 Defining the scope

At first, the group needed help understanding the stakeholder's scope and requirements, which caused a delay in starting the thesis. The group was given some flexibility in defining the scope but needed help to learn the topic entirely. As a result, they spent extra time and effort researching and consulting with professors to gain a better understanding. This helped them define the scope more accurately and align their work accordingly.

Chapter 7

Conclusion

7.1 Introduction

This chapter includes a discussion of the work done, the techniques employed to achieve the intended outcomes, and several constructive suggestions to enhance the quality of the thesis.

7.2 The Work Process

7.2.1 Meetings

The team held meetings with their supervisor, Filip Holik, every week. Although the meetings were typically scheduled for Wednesdays, they had to be postponed or canceled on occasions due to various circumstances. Meeting Minutes with the supervisor can be found in Appendix D.

During the early stages of the project, the team met with the stakeholder every other week. However, as the project progressed, both parties agreed that weekly meetings would be beneficial. In addition, as the demand for help grew, the group required more regular assistance. Meeting Minutes with the stakeholder can be found in Appendix E.

7.2.2 Scrum

Throughout the thesis, the group followed the Scrum framework of agile project management, which required breaking the project into sprints lasting two to four weeks. Daily stand-up meetings were held to keep everyone informed, during which members delivered progress reports on the thesis and discussed plans for the day, including each member's allocated chores. While the group aimed to meet daily, this was sometimes difficult to achieve as some members had work obligations besides their studies.

Despite this, the team had sprint planning and retrospective sessions every two weeks to review progress and establish goals for the next two weeks. During the retrospective meetings, the team engaged in self-reflection by asking questions such as "What should we keep/stop doing?", "What should we do more/less of?" to identify improvement areas for the next sprint cycle. During the sprint planning meetings, on the other hand, the group evaluated completed tasks and discussed what needed to be done before the next sprint period.

Following the Scrum framework helped promote good communication and teamwork among group members. Furthermore, the Kanban board was integrated into the Scrum Framework and proved to be quite helpful in tracking tasks that were ongoing, completed, or yet to be started.

7.2.3 Coordinated schedule

In order to better coordinate their busy schedules, group members with different commitments, such as work and student associations, decided to implement a scheduling method. Doing this allowed them to quickly determine each other's availability and schedule meetings more efficiently. To achieve this, the team utilized the calendar feature within their Teams channel to schedule their activities and workdays.

7.2.4 Draft Submissions

The team set specified deadlines for producing multiple drafts at the start of the project. This approach tried to maintain constant development while avoiding last-minute delays. For instance, the group set an April 3rd target for their first draft, which they could meet. As a result, both the supervisor and the stakeholder received the first

draft on time. After a few days, the team got feedback and proceeded with their work accordingly.
Additionally, the group established a deadline for the final draft. Setting the final draft deadline on the 1st of May, three weeks before the submission date of the 22nd of May, gave the stakeholders and supervisor sufficient time to review the thesis thoroughly. It is essential to mention that the group followed the plan comprehensively, allowing them to send in multiple drafts for review and refinement before the final submission.

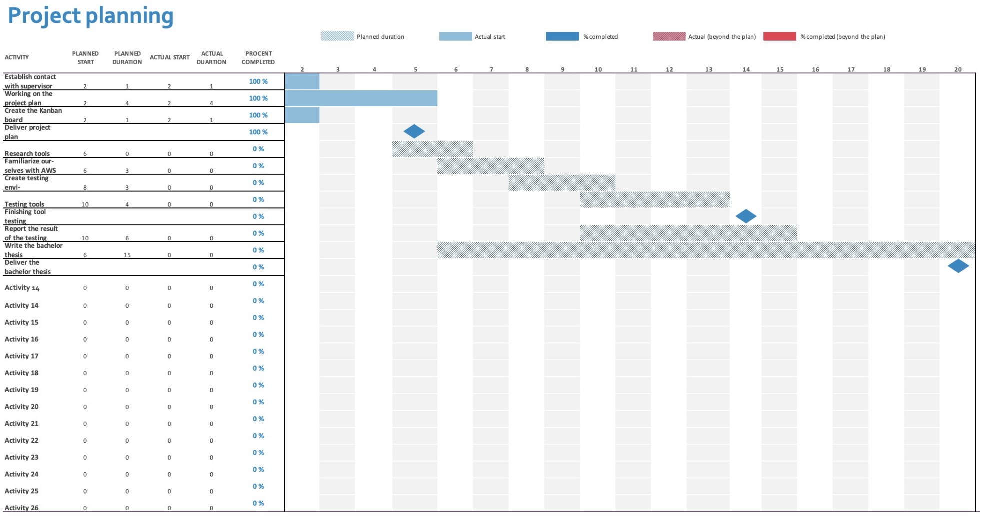
7.2.5 Gantt Chart

Since the group decided to change the scope during the project period, the original Gantt chart could not be followed.

The research on tools was initially considered time-consuming, but with the changes made, this activity became a minor part of the thesis than anticipated. As a result, it took less time than what was first estimated. Furthermore, the “Testing tools” activity was removed since the team wanted to focus less on testing tools and more on integrating testing of the various tools into the pipeline-building process.

In the Gantt chart from the project plan, there was no time set for the pipeline configuration because this part of the scope had not yet been decided. However, the AWS pipeline configuration was included on the new chart, and getting familiar with AWS services and setting up the pipeline was expected to take less time than it did. The documentation for AWS and Terraform is extensive, but navigating it can be overwhelming. With many different services and features offered by AWS, the group had to determine the best fit for their needs. In addition, building the pipeline required a significant amount of functionality to be in place for additional features to work correctly. Therefore, setting up the complete pipeline took an extended amount of time.

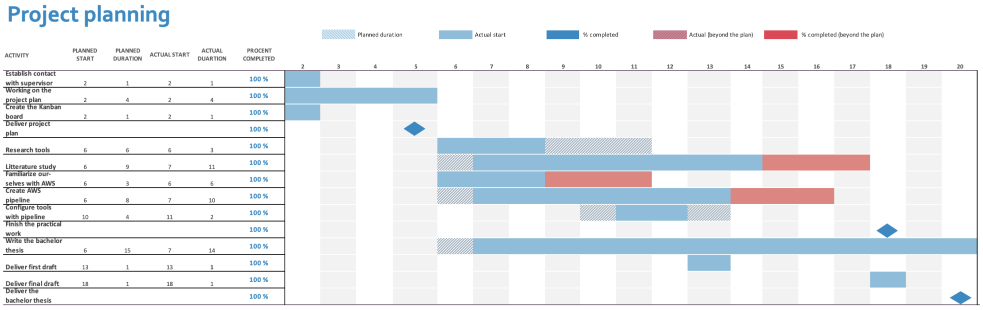
However, the group followed the scheduled deadlines for the various draft submissions. Consequently, the initial draft was submitted on April 3rd in week 13, but after discussions with the supervisor (see meeting minutes in Appendix D.12), it was requested to be rescheduled to May 5th in week 18, resulting in additional work being accomplished within those two days.



Project planning

ACTIVITY	PLANNED START	PLANNED DURATION	ACTUAL START	ACTUAL DUARTION	PROCENT COMPLETED
Establish contact with supervisor	2	1	2	1	100 %
Working on the project plan	2	4	2	4	100 %
Create the Kanban board	2	1	2	1	100 %
Deliver project plan					100 %
Research tools	6	0	0	0	0 %
Familiarize ourselves with AWS	6	3	0	0	0 %
Create testing envi-	8	3	0	0	0 %
Testing tools	10	4	0	0	0 %
Finishing tool testing					0 %
Report the result of the testing	10	6	0	0	0 %
Write the bachelor thesis	6	15	0	0	0 %
Deliver the bachelor thesis					0 %
Activity 14	0	0	0	0	0 %
Activity 14	0	0	0	0	0 %
Activity 15	0	0	0	0	0 %
Activity 16	0	0	0	0	0 %
Activity 17	0	0	0	0	0 %
Activity 18	0	0	0	0	0 %
Activity 19	0	0	0	0	0 %
Activity 20	0	0	0	0	0 %
Activity 21	0	0	0	0	0 %
Activity 22	0	0	0	0	0 %
Activity 23	0	0	0	0	0 %
Activity 24	0	0	0	0	0 %
Activity 25	0	0	0	0	0 %
Activity 26	0	0	0	0	0 %

Figure 7.1: Original Gantt Chart



Project planning

ACTIVITY	PLANNED START	PLANNED DURATION	ACTUAL START	ACTUAL DUARTION	PROCENT COMPLETED
Establish contact with supervisor	2	1	2	1	100 %
Working on the project plan	2	4	2	4	100 %
Create the Kanban board	2	1	2	1	100 %
Deliver project plan					100 %
Research tools	6	6	6	3	100 %
Litterature study	6	9	7	11	100 %
Familiarize ourselves with AWS	6	3	6	6	100 %
Create AWS pipeline	6	8	7	10	100 %
Configure tools with pipeline	10	4	11	2	100 %
Finish the practical work					100 %
Write the bachelor thesis	6	15	7	14	100 %
Deliver first draft	13	1	13	1	100 %
Deliver final draft	18	1	18	1	100 %
Deliver the bachelor thesis					100 %

Figure 7.2: Updated Gantt Chart

7.2.6 Distribution of Work

The group decided to divide the work into two parts at the start of the thesis to ensure that the thesis progressed continuously. The first part is practical, and the second is report writing. The responsibilities were assigned based on each group member's strength and what they most desired to do. As a result, every group member contributed to the thesis, and everyone worked together to complete the report on time.

7.2.7 Goals

P1: *Collaborate effectively with team members to ensure the timely completion of tasks* was achieved. As stated previously, the group incorporated a Kanban board into the Scrum framework allowing for the assignment and tracking of tasks. Additionally, the group utilized various communication platforms, such as Discord and Teams, to ensure effective communication among the group members. Using communication tools already integrated into each member's daily workflow was essential, and these two platforms proved to be the most effective for the group's needs.

P2: *Successfully integrating security tools (e.g., SAST, DAST, SCA) into the SDLC pipeline* was achieved. The group found tools that could be integrated into the pipeline between GitHub and AWS to secure the application code being sent through.

P3: *Implement an automated pipeline using Terraform to build, test, and deploy applications* was achieved. The group created Terraform code that automated the pipeline from the build to the deployment.

R1: *Develop a secure and automated pipeline for the SDLC process using Terraform*, is partially achieved. The group has made significant progress towards achieving a secure pipeline. The group successfully developed a mostly automated and implemented restricted access management measures to enhance pipeline security. However, it is important to acknowledge that a complete assurance of a 100% secure cannot be claimed at this moment in time and will therefore be a part of further work. The is therefore considered to be secure to a certain level, with potential for improvement. Furthermore, the group encountered difficulties in automating the activation of the secret scanner as they were unable to find a method to accomplish this. Nonetheless, the group considered the automation to be unnecessary as it only involved the simple action of pressing a button.

R2: *Produce a report summarizing the project results and recommendations for improving the SDLC pipeline security*, was accomplished. The resulting report provides numerous essential and effective practices that can be implemented to improve security in the SDLC, and the goal was achieved within the requirements given.

7.3 Further Work

For further work, the thesis could be strengthened by performing a broader analysis of various security tools that perform SAST, DAST, and SCA scans - where the selected tools are based on these analyses. During these analyses, an assessment can also be made of which requirements must be met for a tool to be selected.

Including earlier phases in the thesis would have been beneficial to acquire a more thorough grasp of the entire Software Development Life Cycle and adhere to the shift-left methodology, which emphasizes early testing to find vulnerabilities earlier.

Despite successfully automating most of the pipeline, the group encountered challenges in automating Secret Scanning in GitHub. The lack of documentation on enabling it through CLI or Terraform code limited the progress. Consequently, the group prioritized other tasks and allocated more time to this aspect in future work. The ultimate goal remains to achieve a fully automated pipeline.

Maintaining code integrity throughout the pipeline is crucial for a secure system and is, therefore, a part of further work as the group did not manage to sign artifacts. To achieve this, multiple stages should implement the sign and verify process, where artifacts are signed at one stage and then verified at subsequent stages. This reduces the risk of any unauthorized modifications or tampering, which increases the integrity of the data throughout the pipeline.

In addition, to increase code integrity, modifications should not be allowed directly in AWS and the pipeline itself. By implementing this approach, all modifications will be processed through GitHub. Through the commit history, version control is automatically incorporated, and with the utilization of branch protection rules, every code is subject to review.

7.4 Conclusion

The group is pleased to state that they have completed their thesis project, meeting the requirements set by their stakeholder while staying within the project's scope. The input from the stakeholders was critical in determining the group's objectives and requirements, resulting in a successful outcome that met the expectations of everyone involved.

After careful consideration, the group recommends implementing SAST, SCA, and secret scanning during the implementation phase of the Software Development Life Cycle. Additionally, they advise incorporating DAST and manual testing during the testing phase. After deploying the software to production, it is crucial to include security measures during the maintenance phase, such as monitoring and security testing through DAST and SCA scans. To ensure optimal security, it is also recommended to secure the pipeline itself by implementing access control at each stage, branch protection, and commit signing in GitHub. The selection of specific security tools and protection rules should be based on the software's requirements and specifications, with the key aspect being the successful implementation of security measures.

The group hopes that the stakeholder and other organizations reading this thesis will significantly benefit from the thesis, and they highly recommend considering implementing some of their recommendations in their daily work. The group is proud of their accomplishments and hopes this work will encourage more secure development.

Bibliography

[1] NBIM. *Homepage*. 2023. URL: https://www.nbim.no/. (Last accessed: 10.05.2023).

[2] NTB. *Oljefondet utsettes for tre alvorlige dataangrep daglig*. 2022. URL: https://www.digi.no/artikler/oljefondet-utsettes-for-tre-alvorlige-dataangrep-daglig/521643. (Last accessed: 17.01.2023).

[3] Synposys. *Software Development Lifecycle*. 2023. URL: https://www.synopsys.com/glossary/what-is-sdlc.html. (Last accessed: 09.02.2023).

[4] W3schools. *SDLC Iterative Model*. URL: https://www.w3schools.in/sdlc/iterative-model. (Last accessed: 11.05.2023).

[5] Micro Focus. *What Is the SDLC ?* URL: https://www.microfocus.com/en-us/what-is/sdlc. (Last accessed: 13.04.2023).

[6] Sachin R. Doddaguni et al. *Understanding SDLC using CI/CD pipeline*. 2020. URL: https://www.ijsce.org/wp-content/uploads/papers/v9i6/F3405039620.pdf. (Last accessed: 09.05.2023).

[7] Michigan Tech. *Testing Phase in SDLC*. 2016. URL: https://www.mtu.edu/it/security/policies-procedures-guidelines/information-security-program/system-development-lifecycle/. (Last accessed: 21.02.2023).

[8] Noel Ramson. *Testing Phase in SDLC*. 2021. URL: https://study.com/academy/lesson/testing-phase-in-sdlc.html. (Last accessed: 21.02.2023).

[9] Noel Ramson. *Deployment Phase in SDLC*. 2021. URL: https://study.com/academy/lesson/deployment-phase-in-sdlc.html. (Last accessed: 21.02.2023).

[10] Crystal Chilman. *Understanding the Maintenance Phase of the SDLC*. 2023. URL: https://study.com/learn/lesson/maintenance-phase-sdlc-overview-outcomes.html. (Last accessed: 21.02.2023).

[11] Micro Focus. *What is DevSecOps?* URL: https://www.microfocus.com/en-us/what-is/devsecops. (Last accessed: 04.05.2023).

[12] Lu Luo. *Software Testing Techniques.* 2001. URL: https://ignite.org.pk/wp-content/uploads/2018/12/1388051766_rfp1_Software-testing-techniques.pdf. (Last accessed: 24.02.2023).

[13] Kaur Brar Hanmeet and Jai Kaur Puneet. *Differentiating Integration Testing and unit testing.* 2015. URL: https://ieeexplore.ieee.org/abstract/document/7100358. (Last accessed: 27.02.2023).

[14] Ankit Pahuaja. *What is Security Testing and Why is it important?* 2022. URL: https://www.getastra.com/blog/security-audit/what-is-security-testing/. (Last accessed: 22.02.2023).

[15] Imperva. *Application Security Testing.* 2022. URL: https://www.imperva.com/learn/application-security/application-security-testing/. (Last accessed: 28.03.2023).

[16] Thomas Hamilton. *What is BLACK Box Testing? Techniques, Types & Example.* 2023. URL: https://www.guru99.com/black-box-testing.html. (Last accessed: 06.02.2023).

[17] Shubham G. *Black Box Testing: An Important Functional testing Technique.* 2022. URL: https://www.linkedin.com/pulse/black-box-testing-important-functional-technique-shubham-gupta?trk=articles_directory?. (Last accessed: 02.05.2023).

[18] Thomas Hamilton. *White Box Testing – What is, Techniques, Example & Types.* 2023. URL: https://www.guru99.com/white-box-testing.html. (Last accessed: 09.02.2023).

[19] JavaTPoint. *GreyBox Testing.* 2021. URL: https://www.javatpoint.com/grey-box-testing. (Last accessed: 13.04.2023).

[20] Synopsys. *Static Application Security Testing.* 2023. URL: https://www.synopsys.com/glossary/what-is-sast.html. (Last accessed: 11.05.2023).

[21] Snyk. *Static Application Security Testing (SAST).* URL: https://snyk.io/learn/application-security/static-application-security-testing/. (Last accessed: 19.05.2023).

[22] Adam Murray. *SAST – All About Static Application Security Testing.* 2022. URL: https://www.mend.io/resources/blog/sast-static-application-security-testing/. (Last accessed: 12.05.2023).

[23] Synopsys. *Dast.* 2023. URL: https://www.synopsys.com/glossary/what-is-dast.html. (Last accessed: 06.02.2023).

[24] Invicti. *False Positives in Web Application Security – Facing the Challenge.* URL: https://www.invicti.com/white-papers/false-positives-in-application-security-whitepaper/. (Last accessed: 19.05.2023).

[25] Rebecca Warren. *How to Add Application Security Tests to Your CI/CD Pipeline.* 2020. URL: https://www.stackhawk.com/blog/how-to-automate-appsec-testing-in-cicd/. (Last accessed: 11.05.2023).

[26] Synopsys. *Software Composition Analysis.* 2023. URL: https://www.synopsys.com/glossary/what-is-software-composition-analysis.html. (Last accessed: 11.05.2023).

[27] Adam Murray. *Application Security Testing: Security Scanning Vs. Runtime Protection.* 2023. URL: https://www.mend.io/resources/blog/ast-application-security-testing/. (Last accessed: 08.03.2023).

[28] IBM. *Cost of a Data Breach Report 2022.* 2022. URL: https://www.ibm.com/downloads/cas/3R8N1DZJ. (Last accessed: 22.02.2023).

[29] Contributor. *The Cost of Not Building with Security in Mind.* 2016. URL: https://devops.com/cost-not-building-software-security-mind. (Last accessed: 22.02.2023).

[30] Johnson.A et al. *Guide for Security-Focused Configuration Management of Information Systems.* 2011. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-128.pdf. (Last accessed: 28.02.2023).

[31] FIRST. *Common Vulnerability Scoring System SIG.* URL: https://www.first.org/cvss/. (Last accessed: 28.02.2023).

[32] NIST. *Vulnerability Metrics.* URL: https://nvd.nist.gov/vuln-metrics/cvss. (Last accessed: 01.03.2023).

[33] CVE. *Frequently Asked Questions(FAQs).* URL: https://www.cve.org/ResourcesSupport/FAQs#pc_introcve_nvd_relationship. (Last accessed: 01.03.2023).

[34] CWE. *About CWE.* 2022. (Last accessed: 01.03.2023).

[35] Jeff Williams. *OWASP Risk Rating Methodology*. 2023. URL: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. (Last accessed: 06.03.2023).

[36] Nick Barney. *Amazon Web Services (AWS)*. 2022. URL: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services. (Last accessed: 09.02.2023).

[37] Jamie Juviler. *What Is GitHub? (And What Is It Used For?)* 2022. URL: https://blog.hubspot.com/website/what-is-github-used-for. (Last accessed: 09.02.2023).

[38] Srinivas Manepalli. *Building end-to-end AWS DevSecOps CI/CD pipeline with open source SCA, SAST and DAST tools*. 2021. URL: https://aws.amazon.com/blogs/devops/building-end-to-end-aws-devsecops-ci-cd-pipeline-with-open-source-sca-sast-and-dast-tools/. (Last accessed: 05.05.2023).

[39] Microsoft. *What are the Microsoft SDL practices?* 2023. URL: https://www.microsoft.com/en-us/securityengineering/sdl/practices. (Last accessed: 01.05.2023).

[40] Google. *Dependency management*. 2023. URL: https://cloud.google.com/software-supply-chain-security/docs/dependencies. (Last accessed: 25.04.2023).

[41] Cloud Native Computing Foundation. *Software Supply Chain Best Practices*. 2021. URL: https://github.com/cncf/tag-security/blob/main/supply-chain-security/supply-chain-security-paper/CNCF_SSCP_v1.pdf. (Last accessed: 09.03.2023).

[42] Github. *About code scanning*. 2023. URL: https://docs.github.com/en/enterprise-cloud@latest/code-security/secret-scanning/about-secret-scanning#about-secret-scanning. (Last accessed: 21.02.2023).

[43] C.J. May. *Thinking Like a Hacker: Stealing Secrets with a Malicious GitHub Action*. 2022. URL: https://blog.gitguardian.com/thinking-like-a-hacker-stealing-secrets-with-a-malicious-github-action/. (Last accessed: 01.05.2023).

[44] GitGuardian. *The State of Secret Sprawl 2023*. 2023. URL: https://www.gitguardian.com/files/the-state-of-secrets-sprawl-report-2023. (Last accessed: 01.05.2023).

[45] Adam Murray. *Dynamic Application Security Testing: DAST Basics*. 2021. URL: https://www.mend.io/resources/blog/dast-dynamic-application-security-testing/. (Last accessed: 10.03.2023).

[46] Laura Paine. *Defense in Depth: Why You Need DAST, SAST, SCA, and Pen Testing*. 2020. URL: https://www.veracode.com/blog/managing-appsec/defense-depth-why-you-need-dast-sast-sca-and-pen-testing. (Last accessed: 10.03.2023).

[47] Github. *Managing a branch protection rule*. 2023. URL: https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/managing-a-branch-protection-rule. (Last accessed: 10.05.2023).

[48] GitHub. *Access permissions on GitHub*. URL: https://docs.github.com/en/get-started/learning-about-github/access-permissions-on-github. (Last accessed: 13.03.2023).

[49] NIST. *least privilege*. URL: https://csrc.nist.gov/glossary/term/least_privilege. (Last accessed: 13.03.2023).

[50] International Organization for Standardization and International Electrotechnical Commission. *NS-EN ISO/IEC 27002:2022*. 2022. URL: https://www.standard.no/no/Nettbutikk/produktkatalogen/Produktpresentasjon/?ProductID=1450114. (Last accessed: 15.05.2023).

[51] Microsoft. *What is Conditional Access?* 2023. URL: https://learn.microsoft.com/en-us/azure/active-directory/conditional-access/overview. (Last accessed: 15.05.2023).

[52] Nathan Getty, Shaun McCullough and Dave Shackleford. *AppSec/DevSecOps Best Practices in AWS*. 2019. URL: https://d1.awsstatic.com/Marketplace/solutions-center/downloads/AppSec-DevSecOps-AWS-SANS-eBook.pdf. (Last accessed: 01.04.2023).

[53] Cambridge University Press & Assessment. *Framework*. 2023. URL: https://dictionary.cambridge.org/dictionary/english/framework. (Last accessed: 02.05.2023).

[54] SLSA. *About SLSA*. 2023. URL: https://slsa.dev/. (Last accessed: 25.04.2023).

[55] SLSA. *Security levels*. 2023. URL: https://slsa.dev/spec/v1.0/levels. (Last accessed: 25.04.2023).

[56] Murugiah Souppaya, Karen Scarfone and Donna Dodson. *Secure Software Development Framework (SSDF) Version 1.1:Recommendations for Mitigating the Risk of Software Vulnerabilities*. 2022. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf. (Last accessed: 27.04.2023).

[57] Github. *About code scanning with CodeQL*. 2023. URL: https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning-with-codeql. (Last accessed: 08.03.2023).

[58] Github. *About code scanning*. 2023. URL: https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning. (Last accessed: 21.02.2023).

[59] Badsah. *One important feature that Dependabot is missing*. 2022. URL: https://badshah.io/important-dependabot-feature/. (Last accessed: 21.02.2023).

[60] Github. *Kepping your supplychain up to date with Dependabot*. 2023. URL: https://docs.github.com/en/code-security/dependabot. (Last accessed: 21.02.2023).

[61] Stackshare. *Dependabot*. 2023. URL: https://stackshare.io/dependabot. (Last accessed: 20.04.2023).

[62] Regi Publico. *The Pros and Cons of Secret Detection*. 2023. URL: https://globalriskcommunity.com/profiles/blogs/the-pros-and-cons-of-secret-detection. (Last accessed: 05.05.2023).

[63] zaproxy. *Getting started*. URL: https://www.zaproxy.org/getting-started/. (Last accessed: 13.03.2023).

[64] Ivan Homola. *OWASP Zap: 8 Core Features (Pros & Cons)*. 2023. URL: https://www.codiga.io/blog/owasp-zap/. (Last accessed: 19.04.2023).

[65] Github. *About protected branches*. 2023. URL: https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/defining-the-mergeability-of-pull-requests/about-protected-branches#require-pull-request-reviews-before-merging. (Last accessed: 09.03.2023).

[66] SumoLogic. *AWS CodePipeline - definition & overview*. 2023. URL: https://www.sumologic.com/glossary/aws-codepipeline/. (Last accessed: 27.02.2023).

[67] Educative. *What is AWS CodePipeline*. 2023. URL: https://www.educative.io/answers/what-is-the-aws-codepipeline. (Last accessed: 27.02.2023).

[68] AWS. *AWS CodePipeline*. 2023. URL: https://aws.amazon.com/codepipeline/. (Last accessed: 27.02.2023).

[69] AWS. *AWS CodeBuild.* 2023. URL: https://aws.amazon.com/codebuild/. (Last accessed: 01.03.2023).

[70] AWS. *A Build specification reference for CodeBuild.* 2023. URL: https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html. (Last accessed: 01.03.2023).

[71] AWS. *AWS CodeDeploy.* 2023. URL: https://aws.amazon.com/codedeploy/. (Last accessed: 01.03.2023).

[72] AWS. *What is CodeDeploy?* 2023. URL: https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html. (Last accessed: 06.03.2023).

[73] AWS. *What is Amazon S3?* 2023. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. (Last accessed: 06.03.2023).

[74] Amazon Web Services. *What is Amazon EC2?* 2023. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html. (Last accessed: 20.04.2023).

[75] Amazon Web Services. *Instances and AMIs.* 2023. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html. (Last accessed: 11.05.2023).

[76] Hashicorp. *Configuration Syntax.* 2023. URL: https://developer.hashicorp.com/terraform/language/syntax/configuration. (Last accessed: 16.05.2023).

[77] Björn Kimminich. *Why the Juice Shop exists.* 2022. URL: https://pwning.owasp-juice.shop/introduction/motivation.html. (Last accessed: 14.03.2023).

[78] Björn Kimminich. *Codebase 101.* 2022. URL: https://pwning.owasp-juice.shop/part3/codebase.html. (Last accessed: 15.03.2023).

[79] GitHub. *Customizing code scanning.* 2023. URL: https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/customizing-code-scanning. (Last accessed: 28.04.2023).

[80] GitHub. *Find Hard Coded Secrets in Your Code - GitHub Checkout.* 2020. URL: https://www.youtube.com/watch?v=aoL7pDrXt74.

[81] Amazon Web Services. *AWS CodeStar Connections.* 2023. URL: https://docs.aws.amazon.com/codestar-connections/latest/APIReference/Welcome.html. (Last accessed: 01.05.2023).

[82] Amazon Web Services. *AWS CodeBuild concepts.* 2023. URL: https://docs.aws.amazon.com/codebuild/latest/userguide/concepts.html. (Last accessed: 26.04.2023).

[83] Amazon Web Services. *Working with deployment configurations in CodeDeploy.* 2023. URL: https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-configurations.html. (Last accessed: 09.05.2023).

[84] Amazon Web Services. *Working with applications in CodeDeploy.* 2023. URL: https://docs.aws.amazon.com/codedeploy/latest/userguide/applications.html. (Last accessed: 09.05.2023).

[85] Amazon Web Services. *What is Amazon SNS?* 2023. URL: https://docs.aws.amazon.com/sns/latest/dg/welcome.html. (Last accessed: 01.05.2023).

[86] Wiebe de Roos. *The four-eyes principle: what's important in a DevOps world.* 2020. URL: https://amazic.com/the-four-eyes-principle-whats-important-in-a-devops-world/. (Last accessed: 21.03.2023).

[87] zaproxy. *OWASP ZAP.* URL: https://www.zaproxy.org. (Last accessed: 16.05.2023).

[88] Red Hat. *What is Infrastructure as Code (IaC)?* 2022. URL: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac. (Last accessed: 21.04.2023).

[89] Red Hat. *The automated enterprise.* 2022. URL: https://www.redhat.com/rhdc/managed-files/ma-automated-enterprise-e-book-f31879-202209-en.pdf. (Last accessed: 21.04.2023).

[90] GitLab. *What is pipeline as code?* URL: https://about.gitlab.com/topics/ci-cd/pipeline-as-code/. (Last accessed: 11.05.2023).

[91] Jonathan Johnson. *What Is Idempotence?* 2020. URL: https://www.bmc.com/blogs/idempotence/. (Last accessed: 21.04.2023).

[92] Kelsey Hightower. *The Secure Software Supply Chain by Kelsey Hightower (Strange Loop 2022).* 2022. URL: https://www.youtube.com/watch?v=JC-xCXcyNXI&ab_channel=StrangeLoopConference. (Last accessed: 16.05.2023).

[93] Hala Assal and Sonia Chiasson. *Security in the Software Development Lifecycle.* 2018. URL: https://www.usenix.org/system/files/conference/soups2018/soups2018-assal.pdf. (Last accessed: 20.04.2023).

[94] divyanshu_gupta1. *Software Development Life Cycle (SDLC)*. 2021. URL: https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/. (Last accessed: 17.01.2023).

[95] Digite. *What Is Scrum Methodology? & Scrum Project Management.* 2023. URL: https://www.digite.com/agile/scrum-methodology/. (Last accessed: 25.01.2023).

[96] Dave West. *Agile scrum roles and responsibilities.* 2023. URL: https://www.atlassian.com/agile/scrum/roles. (Last accessed: 25.01.2023).

[97] NTNU. *Overleaf.* 2023. URL: https://i.ntnu.no/wiki/-/wiki/Norsk/Overleaf. (Last accessed: 17.01.2023).

[98] NTNU. *Matrix for risk assessments at NTNU.* 2013. URL: https://www.ntnu.no/hms/retningslinjer/HMSRV2604_Risikomatrise_100209.pdf. (Last accessed: 17.01.2023).

Appendix A

Results of OWASP ZAP scan

This appendix includes all the output from the DAST scan using OWASP ZAP.

A.1 ZAP scan CLI output

```
Using the Automation Framework
Total of 112 URLs
PASS: Vulnerable JS Library (Powered by Retire.js) [10003]
PASS: In Page Banner Information Leak [10009]
PASS: Cookie No HttpOnly Flag [10010]
PASS: Cookie Without Secure Flag [10011]
PASS: Re-examine Cache-control Directives [10015]
PASS: Content-Type Header Missing [10019]
PASS: Anti-clickjacking Header [10020]
PASS: X-Content-Type-Options Header Missing [10021]
PASS: Information Disclosure - Debug Error Messages [10023]
PASS: Information Disclosure - Sensitive Information in URL [10024]
PASS: Information Disclosure - Sensitive Information in HTTP Referrer
    ↪ Header [10025]
PASS: HTTP Parameter Override [10026]
PASS: Open Redirect [10028]
PASS: Cookie Poisoning [10029]
PASS: User Controllable Charset [10030]
PASS: User Controllable HTML Element Attribute (Potential XSS) [10031]
PASS: Viewstate [10032]
PASS: Directory Browsing [10033]
PASS: Heartbleed OpenSSL Vulnerability (Indicative) [10034]
PASS: Strict-Transport-Security Header [10035]
PASS: HTTP Server Response Header [10036]
PASS: Server Leaks Information via "X-Powered-By" HTTP Response Header
    ↪  Field(s) [10037]
PASS: X-Backend-Server Header Information Leak [10039]
PASS: Secure Pages Include Mixed Content [10040]
PASS: HTTP to HTTPS Insecure Transition in Form Post [10041]
PASS: HTTPS to HTTP Insecure Transition in Form Post [10042]
PASS: User Controllable JavaScript Event (XSS) [10043]
```

```
PASS: Big Redirect Detected (Potential Sensitive Information Leak)
    ↪ [10044]
PASS: Retrieved from Cache [10050]
PASS: X-ChromeLogger-Data (XCOLD) Header Information Leak [10052]
PASS: Cookie without SameSite Attribute [10054]
PASS: CSP [10055]
PASS: X-Debug-Token Information Leak [10056]
PASS: Username Hash Found [10057]
PASS: X-AspNet-Version Response Header [10061]
PASS: PII Disclosure [10062]
PASS: Hash Disclosure [10097]
PASS: Weak Authentication Method [10105]
PASS: Reverse Tabnabbing [10108]
PASS: Absence of Anti-CSRF Tokens [10202]
PASS: Private IP Disclosure [2]
PASS: Session ID in URL Rewrite [3]
PASS: Script Passive Scan Rules [50001]
PASS: Stats Passive Scan Rule [50003]
PASS: Insecure JSF ViewState [90001]
PASS: Java Serialization Object [90002]
PASS: Sub Resource Integrity Attribute Missing [90003]
PASS: Charset Mismatch [90011]
PASS: Application Error Disclosure [90022]
PASS: WSDL File Detection [90030]
PASS: Loosely Scoped Cookie [90033]
WARN-NEW: Cross-Domain JavaScript Source File Inclusion [10017] x 10
       http://172.16.10.100:3000/ (200 OK)
       http://172.16.10.100:3000/ (200 OK)
       http://172.16.10.100:3000/home/build/routes/fileServer.js:16:13
           ↪  (200 OK)
       http://172.16.10.100:3000/home/build/routes/fileServer.js:16:13
           ↪  (200 OK)
       http://172.16.10.100:3000/home/build/routes/fileServer.js:32:18
           ↪  (200 OK)
WARN-NEW: Information Disclosure - Suspicious Comments [10027] x 2
       http://172.16.10.100:3000/main.js (200 OK)
       http://172.16.10.100:3000/vendor.js (200 OK)
WARN-NEW: Content Security Policy (CSP) Header Not Set [10038] x 11
       http://172.16.10.100:3000/ (200 OK)
       http://172.16.10.100:3000/ftp (200 OK)
       http://172.16.10.100:3000/ftp/coupons_2013.md.bak (403
           ↪ Forbidden)
       http://172.16.10.100:3000/ftp/eastere.gg (403 Forbidden)
       http://172.16.10.100:3000/ftp/encrypt.pyc (403 Forbidden)
WARN-NEW: Non-Storable Content [10049] x 11
       http://172.16.10.100:3000/ftp/eastere.gg (403 Forbidden)
       http://172.16.10.100:3000/robots.txt (200 OK)
       http://172.16.10.100:3000/ (200 OK)
       http://172.16.10.100:3000/assets/public/favicon_js.ico (200 OK)
       http://172.16.10.100:3000/ftp/acquisitions.md (200 OK)
WARN-NEW: Deprecated Feature Policy Header Set [10063] x 11
       http://172.16.10.100:3000/ (200 OK)
       http://172.16.10.100:3000/ftp (200 OK)
       http://172.16.10.100:3000/ftp/coupons_2013.md.bak (403
           ↪ Forbidden)
       http://172.16.10.100:3000/ftp/eastere.gg (403 Forbidden)
       http://172.16.10.100:3000/ftp/encrypt.pyc (403 Forbidden)
WARN-NEW: Timestamp Disclosure - Unix [10096] x 1
       http://172.16.10.100:3000/main.js (200 OK)
WARN-NEW: Cross-Domain Misconfiguration [10098] x 11
       http://172.16.10.100:3000/ (200 OK)
       http://172.16.10.100:3000/assets/public/favicon_js.ico (200 OK)
       http://172.16.10.100:3000/ftp (200 OK)
       http://172.16.10.100:3000/ftp/acquisitions.md (200 OK)
       http://172.16.10.100:3000/main.js (200 OK)
```

```
WARN-NEW: Modern Web Application [10109] x 13
        http://172.16.10.100:3000/ (200 OK)
        http://172.16.10.100:3000/home/build/routes/fileServer.js:16:13
            ↪  (200 OK)
        http://172.16.10.100:3000/home/build/routes/fileServer.js:32:18
            ↪  (200 OK)
        http://172.16.10.100:3000/home/build/routes/runtime.js (200 OK)
        http://172.16.10.100:3000/home/node_modules/express/lib/router/
            ↪ index.js:280:10 (200 OK)
WARN-NEW: Dangerous JS Functions [10110] x 2
        http://172.16.10.100:3000/main.js (200 OK)
        http://172.16.10.100:3000/vendor.js (200 OK)
FAIL-NEW: 0 FAIL-INPROG: 0 WARN-NEW: 9 WARN-INPROG: 0 INFO: 0 IGNORE:
    ↪ 0 PASS: 51
```

Code A.1: OWASP ZAP baseline scan

A.2 ZAP scan PDF report

ZAP Scanning Report

Site: http://172.16.10.100:3000

Generated on Thu, 18 May 2023 17:11:02

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	4
Informational	4
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	11
Cross-Domain Misconfiguration	Medium	11
Cross-Domain JavaScript Source File Inclusion	Low	10
Dangerous JS Functions	Low	2
Deprecated Feature Policy Header Set	Low	12
Timestamp Disclosure - Unix	Low	1
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	11
Storable and Cacheable Content	Informational	2
Storable but Non-Cacheable Content	Informational	10

Alert Detail

Medium	**Content Security Policy (CSP) Header Not Set**
Description	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audioand video files.
URL	http://172.16.10.100:3000/
Method	GET

Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/ftp
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/ftp/coupons_2013.md.bak
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/ftp/eastere.gg
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/ftp/encrypt.pyc
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/ftp/package.json.bak
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/ftp/suspicious_errors.yml
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:280:10
Method	GET

Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:328:13
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:365:14
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/sitemap.xml
Method	GET
Parameter	
Attack	
Evidence	
Instances	11
Solution	Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.
Reference	https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html http://www.w3.org/TR/CSP/ http://w3c.github.io/webappsec/specs/content-security-policy/csp-specification.dev.html http://www.html5rocks.com/en/tutorials/security/content-security-policy/ http://caniuse.com/#feat=contentsecuritypolicy http://content-security-policy.com/
CWE Id	693
WASC Id	15
Plugin Id	10038
Medium	**Cross-Domain Misconfiguration**
Description	Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server
URL	http://172.16.10.100:3000/
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *

URL	http://172.16.10.100:3000/assets/public/favicon_js.ico
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/ftp
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/ftp/acquisitions.md
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/main.js
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/polyfills.js
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/robots.txt
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/runtime.js
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/sitemap.xml

Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/styles.css
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
URL	http://172.16.10.100:3000/vendor.js
Method	GET
Parameter	
Attack	
Evidence	Access-Control-Allow-Origin: *
Instances	11
Solution	Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance). Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allowthe web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.
Reference	https://vulncat.fortify.com/en/detail?id=desc.config.dotnet.html5_overly_permissive_cors_policy
CWE Id	264
WASC Id	14
Plugin Id	10098
Low	**Cross-Domain JavaScript Source File Inclusion**
Description	The page includes one or more script files from a third-party domain.
URL	http://172.16.10.100:3000/
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
URL	http://172.16.10.100:3000/
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:280:10

Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:280:10
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:328:13
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:328:13
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:365:14
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:365:14
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
URL	http://172.16.10.100:3000/sitemap.xml
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>

URL	http://172.16.10.100:3000/sitemap.xml
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
Instances	10
Solution	Ensure JavaScript source files are loaded from only trusted sources, and the sources can't be controlled by end users of the application.
Reference	
CWE Id	829
WASC Id	15
Plugin Id	10017

Low	**Dangerous JS Functions**
Description	A dangerous JS function seems to be in use that would leave the site vulnerable.
URL	http://172.16.10.100:3000/main.js
Method	GET
Parameter	
Attack	
Evidence	bypassSecurityTrustHtml
URL	http://172.16.10.100:3000/vendor.js
Method	GET
Parameter	
Attack	
Evidence	bypassSecurityTrustHtml
Instances	2
Solution	See the references for security advice on the use of these functions.
Reference	https://angular.io/guide/security
CWE Id	749
WASC Id	
Plugin Id	10110

Low	**Deprecated Feature Policy Header Set**
Description	The header has now been renamed to Permissions-Policy.
URL	http://172.16.10.100:3000/
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy

URL	http://172.16.10.100:3000/ftp
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/ftp/coupons_2013.md.bak
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/ftp/eastere.gg
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/ftp/encrypt.pyc
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/ftp/package.json.bak
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/ftp/suspicious_errors.yml
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/main.js
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/polyfills.js

Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/runtime.js
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/sitemap.xml
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
URL	http://172.16.10.100:3000/vendor.js
Method	GET
Parameter	
Attack	
Evidence	Feature-Policy
Instances	12
Solution	Ensure that your web server, application server, load balancer, etc. is configured to set the Permissions-Policy header instead of the Feature-Policyheader.
Reference	https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Feature-Policy https://scotthelme.co.uk/goodbye-feature-policy-and-hello-permissions-policy/
CWE Id	16
WASC Id	15
Plugin Id	10063

Low	**Timestamp Disclosure - Unix**
Description	A timestamp was disclosed by the application/web server - Unix
URL	http://172.16.10.100:3000/main.js
Method	GET
Parameter	
Attack	
Evidence	1734944650
Instances	1
Solution	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.
Reference	http://projects.webappsec.org/w/page/13246936/Information%20Leakage

CWE Id	200
WASC Id	13
Plugin Id	10096

Informational	**Information Disclosure - Suspicious Comments**
Description	The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.
URL	http://172.16.10.100:3000/main.js
Method	GET
Parameter	
Attack	
Evidence	query
URL	http://172.16.10.100:3000/vendor.js
Method	GET
Parameter	
Attack	
Evidence	query
Instances	2
Solution	Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.
Reference	
CWE Id	200
WASC Id	13
Plugin Id	10027

Informational	**Modern Web Application**
Description	The application appears to be a modern web application. If you need to explore it automatically then the Ajax Spider may well be more effective than the standard one.
URL	http://172.16.10.100:3000/
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
URL	http://172.16.10.100:3000/home/build/routes/fileServer.js:16:13
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
URL	http://172.16.10.100:3000/home/build/routes/fileServer.js:32:18

Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"> </script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:280:10
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"> </script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:328:13
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"> </script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:365:14
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"> </script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/index.js:376:14
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"> </script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/layer.js:95:5
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"> </script>
URL	http://172.16.10.100:3000/home/node_modules/express/lib/router/styles.css
Method	GET
Parameter	
Attack	

Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
URL	http://172.16.10.100:3000/home/node_modules/serve-index/index.js:145:39
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
URL	http://172.16.10.100:3000/sitemap.xml
Method	GET
Parameter	
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
Instances	11
Solution	This is an informational alert and so no changes are required.
Reference	
CWE Id	
WASC Id	
Plugin Id	10109

Informational	**Storable and Cacheable Content**
Description	The response contents are storable by caching components such as proxy servers, and may be retrieved directly from the cache, rather than from the origin server by the caching servers, in response to similar requests from other users. If the response data is sensitive, personal or user-specific, this may result in sensitive information being leaked. In somecases, this may even result in a user gaining complete control of the session of another user, depending on the configuration of the caching components in use in their environment. This is primarily an issue where "shared" caching servers such as "proxy" caches are configured on the local network. This configuration is typically found in corporate or educational environments, for instance.
URL	http://172.16.10.100:3000/ftp
Method	GET
Parameter	
Attack	
Evidence	
URL	http://172.16.10.100:3000/robots.txt
Method	GET
Parameter	
Attack	
Evidence	
Instances	2

Solution	Validate that the response does not contain sensitive, personal or user-specific information. If it does, consider the use of the following HTTP response headers, to limit, or prevent the content being stored and retrieved from the cache by another user: Cache-Control: no-cache, no-store, must-revalidate, private Pragma: no-cache Expires: 0 This configuration directs both HTTP 1.0 and HTTP 1.1 compliant caching servers to not store the response, and to not retrieve the response (without validation) from the cache, in response to a similar request.
Reference	https://tools.ietf.org/html/rfc7234 https://tools.ietf.org/html/rfc7231 http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html (obsoleted by rfc7234)
CWE Id	524
WASC Id	13
Plugin Id	10049

Informational	**Storable but Non-Cacheable Content**
Description	The response contents are storable by caching components such as proxy servers, but will not be retrieved directly from the cache, without validatingthe request upstream, in response to similar requests from other users.
URL	http://172.16.10.100:3000/
Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/assets/public/favicon_js.ico
Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/ftp/acquisitions.md
Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/ftp/legal.md
Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/main.js

Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/polyfills.js
Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/runtime.js
Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/sitemap.xml
Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/styles.css
Method	GET
Parameter	
Attack	
Evidence	max-age=0
URL	http://172.16.10.100:3000/vendor.js
Method	GET
Parameter	
Attack	
Evidence	max-age=0
Instances	10
Solution	
Reference	https://tools.ietf.org/html/rfc7234 https://tools.ietf.org/html/rfc7231 http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html (obsoleted by rfc7234)
CWE Id	524
WASC Id	13
Plugin Id	10049

Appendix B

Project Plan

B.1 Introduction

The group has together created a project plan that will be highly relevant throughout the entire bachelor process. The report contains scope, goals, routines and roles as well as other important topics that is well needed to be able to work properly towards the end-goal. If the group gets lost along the way or something becomes unclear, this project plan will work as a guideline on how the group is going to work together.

B.2 Goals and restrictions

B.2.1 Background

Norwegian Bank Investment Management, from now on referred to as NBIM, is a division within the central bank responsible for overseeing the Government Pension Fund of Norway, which has a worth of 13,000 billion Norwegian kroner [1]. Due to its large value, the fund is a major target for potential malicious actors. It faces an average of three severe cyber attacks per day, totaling around 100,000 attacks each year. Out of these, more than 1,000 are considered significant threats [2]. Therefore, it is crucial for NBIM, as well as other organizations, to ensure the security of their systems and applications before deploying them into their cloud services.

Software Development Life Cycle (SDLC), describes how software applications are built - from planning through implementation and running in production. It also includes ensuring security at the different stages of software development. In order to accommodate frequent deployments to production, it is important to automate the security testing by building it into the deployment pipeline. The security testing can further benefit from shift-left, where testing is done as early as possible in the pipeline. Given that source code can be accessed by anyone, it is important to consider potential vulnerabilities during the development process. Implementing a strong and secure software development life cycle is essential to prevent attacks from hackers and other malicious actors on your application [3]. Securing the SDLC is a large and actively developed area with a lot of interest from the industry. Demonstrating the integration and practical application of various tools and methods can be beneficial for both NBIM and other organizations.

B.2.2 Project goals

Our project goals will be separated into three different categories; performance goals, result goals and learning goals. Performance goals are the targets that the group sets for themselves in the long-term, with the aim of achieving a desired level of performance or outcome. Result goals are focused on the specific result or outcomes that the group aims to achieve at the end of the project. These goals outline what the final product will be and the value it will provide to the stakeholder. Learning goals are the knowledge that is wished to acquire during the project, and what new skills the group want to be left with, and after the project ends - the acquired knowledge and skills are intended to be retained and used in the future.

B.2.2.1 Performance goals

- The group will be working towards achieving the best possible result, both for the stakeholder and for themselves.
- The group will be as efficient as possible during the work-hours. Which are from 10-14. However, the remaining hours group members are allowed to work on different aspects of the thesis were it is mandatory for a bit in-depth work, but that is not crucial to have done at that very time.
- The group will work towards good cooperation, making the process pleasurable for all individuals involved.

B.2.2.2 Result goals

- To finish a report which can be used for securing deployment of software for companies, both NBIM and others.
- Achieve a grade that the group is satisfied with.
- To create a product the members can use in future work.
- Enhance efficiency for securing deployment of software for everyone involved in developing software, regardless of previous knowledge.

B.2.2.3 Learning goals

- After finishing the thesis, the group hopes to have a better understanding about SDLC and securing the different stages of this deployment cycle.
- Acquire knowledge that improves future work for the group members.
- To learn more about project management and working in teams.

B.2.3 Framework

B.2.3.1 Time frame

- The project plan has to be delivered and signed within the 31th of January 2023.

- The finished bachelor thesis is to be delivered by the 22nd of May 2023.

B.3 Scope

B.3.1 Problem

Create a report outlining how to best secure parts of the SDLC. We want to focus on the deployment pipeline, from submitting new code to GitHub to deploying it to Amazon Web Services. We will review different tools, as well as implementing a proof of concept demonstrating how the different tools can be used together. The proof of concept should demonstrate how we can maintain integrity of the code throughout the pipeline, as well as scanning for security misconfigurations and vulnerabilities at a key stage of the pipeline. The user experience and ability to scale an enterprise environment should be taken into consideration.

B.3.2 Problem delimitation

The primary objective of tool evaluation is to assess the most widely utilized tools in order to ensure applicability for a majority of users. The allocated budget provided by stakeholders must be adhered to when procuring licenses and other necessary technologies for the production of a comprehensive report.

In addition the group will focus primarily on step five (Product Testing and Integration) and six (Deployment and Maintenance Of Product) of the SDLC, but if desired it can be necessary to look at step four (Developing Product). This is because it seems the most necessary to be exploring tools related to deployment pipelines, and thus will not focus on the earlier stages of the SDLC since this is outside the scope and wishes of NBIM. [92] [94]

B.4 Project organization

B.4.1 Roles and area of responsibility

Group leader

The group leader holds the responsibility for ensuring the participation and collaboration of all group members in the completion of the project. In the event of interpersonal conflicts within the group, it is the duty of the leader to mediate and resolve any issues that may arise.

The group leader is: Thea Urne

Head of communication

The head of communication is responsible for all external communication, which consist of all contact between external business and supervisor provided by Norwegian University of Science and Technology Gjøvik.

The Head of Communication serves as the primary spokesperson during formal meetings with stakeholder and supervisor, and is responsible for creating agendas for these meetings. In contrast, informal and shorter meetings (5-15 minutes) will be conducted as a collective effort, where all team members are given an opportunity to voice their contributions.

The Head of Communication is: Celina Heimdal Brynildsen

Head of documentation

The Head of Documentation is responsible for making sure all documents are in place and structured correctly. There will be documentations like meeting minutes, time sheets, logs and reports, and it is therefore important for one to have full overview.

The head of documentation is: Anniken Arildset

Secretary

The secretary will be responsible for writing reports from meetings that the group will have with external business and supervisor. The secretary will then add all reports to a folder created in Teams, where the head of documentation will make sure that it has been done correctly.

Secretary is: Thea Urne

Head of sources

The head of sources role will be divided between two people, where they will control that sources are written properly and that the group follow the right structure.

The head of sources will be: Sebastian Hestsveen and Celina Heimdal Brynildsen

Head of quality assurance

The role of Head of Quality Assurance will be shared by two individuals, who will conduct weekly reviews of all written materials to ensure that all typographical errors have been corrected and the structure of the text adheres to the established guidelines of the group.

The head of quality assurance will be: Anniken Arildset and Thea Urne.

B.4.2 Routines

- Meetings with the supervisor will be planned weekly.
- Every other Wednesday, the group will conduct the sprint retrospective.
- Meetings with the stakeholder will take place every other Thursday at 12pm. However, if both the group and the stakeholder decides that a meeting isn't necessary, these meetings can be cancelled. These meetings will be the sprint review.
- The group will have a meeting every Friday at 10am where the week will be summarized, and the upcoming week will be planned. Every other week this will be the sprint planning meeting. This is recommended to have in the beginning of the week, though because of work, the group found the best solution to this being Fridays.

- Hours will be registered in Excel and Word documents, each group member needs to check that the lists are updated by the end of the week.
- It is expected that group members are available from 10-14 Wednesday, Thursday and Friday. It is expected that group members are on campus during these hours, but after that group members can sit wherever they want.
- It is expected that all group members work at least 30 hours a week unless there is a valid reason for why they haven't worked their hours.
- Teams, Discord and Outlook will be the groups primarily communications platforms both internally and externally.

B.4.3 Group rules

- If a group member does not show up when expected they must buy a bag of Gifflar each to the rest of the group members. If a group member show up after 30 minutes they must buy a bag of Gifflar, this will increase every thirty minutes, meaning if they are 1,5 hours late they must buy three bags of Gifflar. This will end at 1,5 hours, so if a group member is more than 1,5 hours late they only need to buy three bags of Gifflar.
- If one of the group members is late, it is obligatory to report this to the rest of the group beforehand, so that work sessions and meetings can be reorganized related to this. It is expected that group members have a valid reason if they cannot attend meetings and work sessions. The group members must notify the group at least one day in advance for their absence to be valid. Acute sickness can be notified the same day as a meeting or work sessions.
- The given task to each member has to be completed in time, if this is not possible, the rest of the group must be notified in advance.
- If a conflict arises, the group will firstly try to handle it internally. If the group does not see eye to eye, supervisor will be contacted.
- If discussions occurs were the group have to vote, and the alternatives have equal amount of votes, the group leader will have an extra vote. Other than this extraordinary event, all members of the group have equally one vote each.

B.5 Planning, follow-up and reporting

B.5.1 Project Management Methodology

The group discussed four types of development methodologies; Scrum, Kanban, Waterfall and Scientific method. Each group member made a presentation about their chosen methodology, and their strengths and weaknesses. After all group members had their presentation, the group then decided to adopt Scrum as our project management approach because it allows us to adapt and improve our work based on feedback from supervisor and our stakeholder. It is an iterative methodology and has focus on continuous improvement, unlike the waterfall method which does not allow for adjustments once a stage is completed. Additionally, the group will be implementing a Kanban board to provide a visual representation of the project's progress and tasks. An application in Teams will be used to create the Kanban board. Because Scrum is being used, it is necessary to have daily stand-up meetings. These stand-up meetings will also help keep team members informed and on track.

B.5.2 Scrum

Scrum is an agile methodology that allows for continuous improvement and adjustments through the whole development process [95]. By having regular meetings within the group and with the stakeholder, everyone involved is kept up to date on what is done and what will be done in the future. The group will have short, daily meetings lasting about 15 minutes. During these meetings, the participants will discuss what was done the day before, what will be done today and any obstacles that may prevent the members from finishing their tasks.

The work will be split into two week sprints. Every sprint will be planned in a sprint planning meeting. After every sprint, there will be a sprint review with the stakeholder. Sprint reviews will be an assessment of the work done according to the product requirements. In addition to the sprint reviews, the group will have sprint retrospectives. These meetings will be a discussion among the group members about the efficiency and cooperation during the sprint. The members will assess improvements for productivity and bettering the process for the next sprint.

During this process, there are three main Scrum roles: product owner, scrum master and team. The product owner is responsible for knowing the customers and setting product requirements. In this case, this will be the stakeholder. The scrum master is responsible for making sure the Scrum method is done properly and that the sprints are properly communicated to the product owner. The team consists of workers with the appropriate knowledge and skills to complete the task [96].

- Product owner - NBIM
- Scrum master - Celina Heimdal Brynildsen
- Team - Anniken Arildset, Sebastian Hestsveen and Thea Urne

B.5.3 Follow-up

- The group will have follow-up meetings every Friday were it is expected that everything that has been done that week and what needs to be done for the week to come is the main agenda.
- The group will also have 15 minutes meetings every day so that all group members will have the opportunity to share their ongoing work with group members and ask for help if it is necessary.
- As a part of the Scrum methodology, the group will every other week have a meeting with the stakeholder were it is expected to go through what the group have done and give the stakeholder the opportunity to give feedback on what has been done. After this, the group will then have a private meeting with each other to go through the feedback that was received.

B.6 Organization of quality assurance

B.6.1 Documentation

All documentation, including but not limited to notes, time sheets and meeting minutes, will be saved and shared through Teams. All members and the supervisor have access to these through our shared channel. By using Teams to store documents, everyone can easily see each other's work and collaborate.

On the group's Friday meetings, there will be taken a backup of all major reports, like the project plan and bachelor thesis. This will be done by uploading the files to Google Drive. It will also be taken daily backup of the reports, and these will be uploaded to GitLab.

Smaller documentation, like notes, will mainly be done using Microsoft Word, alternatively other tools Microsoft provide. The major report on the other hand, will be written using Overleaf. Overleaf is an online LaTeX editor that allows all members to co-write on the same document simultaneously [97].

B.6.2 Plan for testing and inspection

In the time of writing, the group does not have enough information regarding this topic, and will therefore expand this topic in later on.

B.6.3 Risk analysis

The following presents a general risk analysis for our project, identifying potential problems that may arise and assigning them a probability and consequence value. It also includes measures to address issues that may come up. The analysis is divided into three categories: green (acceptable risk), yellow (moderate risk), and red (unacceptable risk).

Consequence	Catastrophic	🟨	🟥	🟥	🟥	🟥
	Major	🟨	🟨 2	🟥 1	🟥	🟥
	Moderate	🟨	🟨 3, 4, 5	🟨	🟥	🟥
	Minor	🟩	🟩	🟨	🟨	🟥
	Insignificant	🟩	🟩	🟩	🟨	🟨
		Rare	Unlikely	Possible	Likely	Certain
Probability						

Table B.1: Risk matrix
[98]

Risk scenario 1:

Risk scenario	Stretching beyond scope
Description	Due to poor planing and communication in the group, the group have taken on too much and this leads to a larger scope. This leads to that the group cannot deliver on the original task
Probability	Possible
Consequence	Major
Overall risk	🟥 Very serious

Table B.2: Scenario 1

Measures:
By implementing proper oversight and developing a solid project plan will result in minimizing potential harm. Additionally, before making any decisions to expand the project, it is important to thoroughly assess the necessity of doing so in order to prevent scope creep.

Risk scenario 2:

Risk scenario	Loss of data
Description	The group will lose important data or documents
Probability	Unlikely
Consequence	Major
Overall risk	🟨 Moderate

Table B.3: Scenario 2

Measures:
To mitigate this risk, the group need to ensure that our data is stored in multiple locations and devices. Regular backups should also be performed to these locations to minimize loss in the event of an incident.

Risk scenario 3:

Risk scenario	Group conflict
Description	Group will disagree
Probability	Unlikely
Consequence	Moderate
Overall risk	Moderate

Table B.4: Scenario 3

Measures:
Encourage open communication among group members and actively listen to each other's perspectives. Identify the root causes of the conflict and work to address them directly. If the conflict cannot be resolved within the group, consider seeking assistance from a neutral counselor.

Risk scenario 4:

Risk scenario	Loss of contact with stakeholder
description	For some reason we cannot reach the stakeholder and they do not respond to us
Probability	Unlikely
Consequence	Moderate
Overall risk	Moderate

Table B.5: Scenario 4

Measures:
Initially, the group would attempt to contact individuals through email. If this is unsuccessful, Celina, one of the group members, will contact the stakeholder at work since she works at NBIM. In the worst case the group could finish this task without them since we don't need access to their systems.

Risk scenario 5:

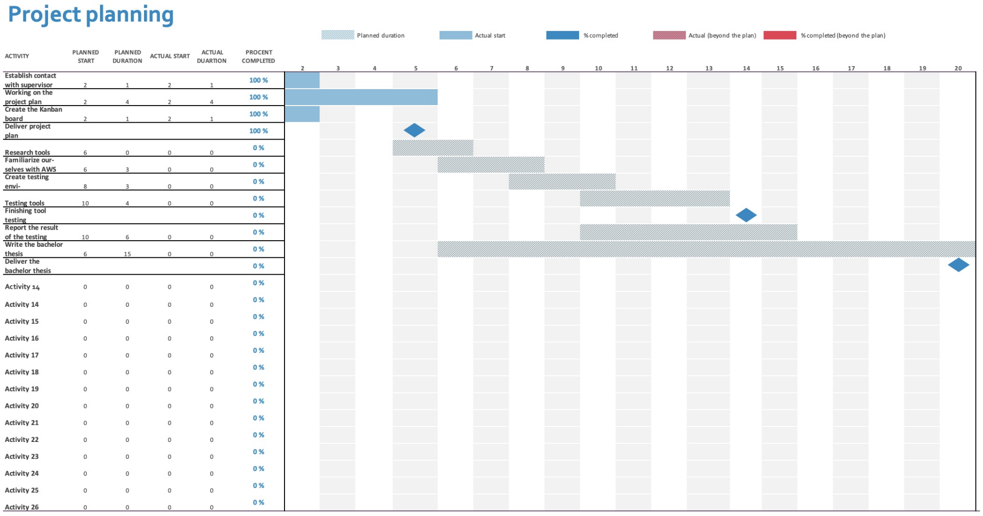
Risk scenario	A group member gets sick
Description	Due to a new pandemic or other circumstances, a group member may become ill and unable to contribute to the group's efforts, resulting in the group having to redistribute the workload among fewer members.
Probability	Unlikely
Consequence	Moderate
Overall risk	Moderate

Table B.6: Scenario 5

Measures:
It is difficult prevent someone from becoming sick, however our effective documentation and god communication allow for continuation of work by others in the event that a team member falls ill.

B.7 Plan for execution

B.7.1 Gantt chart



Project planning

ACTIVITY	PLANNED START	PLANNED DURATION	ACTUAL START	ACTUAL DUARTION	PROCENT COMPLETED
Establish contact with supervisor	2	1	2	1	100 %
Working on the project plan	2	4	2	4	100 %
Create the Kanban board	2	1	2	1	100 %
Deliver project plan					100 %
Research tools	6	0	0	0	0 %
Familiarize ourselves with AWS	6	3	0	0	0 %
Create testing envi-	8	3	0	0	0 %
Testing tools	10	4	0	0	0 %
Finishing tool testing					0 %
Report the result of the testing	10	6	0	0	0 %
Write the bachelor thesis	6	15	0	0	0 %
Deliver the bachelor thesis					0 %
Activity 14	0	0	0	0	0 %
Activity 14	0	0	0	0	0 %
Activity 15	0	0	0	0	0 %
Activity 16	0	0	0	0	0 %
Activity 17	0	0	0	0	0 %
Activity 18	0	0	0	0	0 %
Activity 19	0	0	0	0	0 %
Activity 20	0	0	0	0	0 %
Activity 21	0	0	0	0	0 %
Activity 22	0	0	0	0	0 %
Activity 23	0	0	0	0	0 %
Activity 24	0	0	0	0	0 %
Activity 25	0	0	0	0	0 %
Activity 26	0	0	0	0	0 %

B.8 Signature

Approved: ______________________________

Anniken Arildset

Approved: ______________________________

Celina Heimdal Brynildsen

Approved: ______________________________

Sebastian Hestsveen

Approved: ______________________________

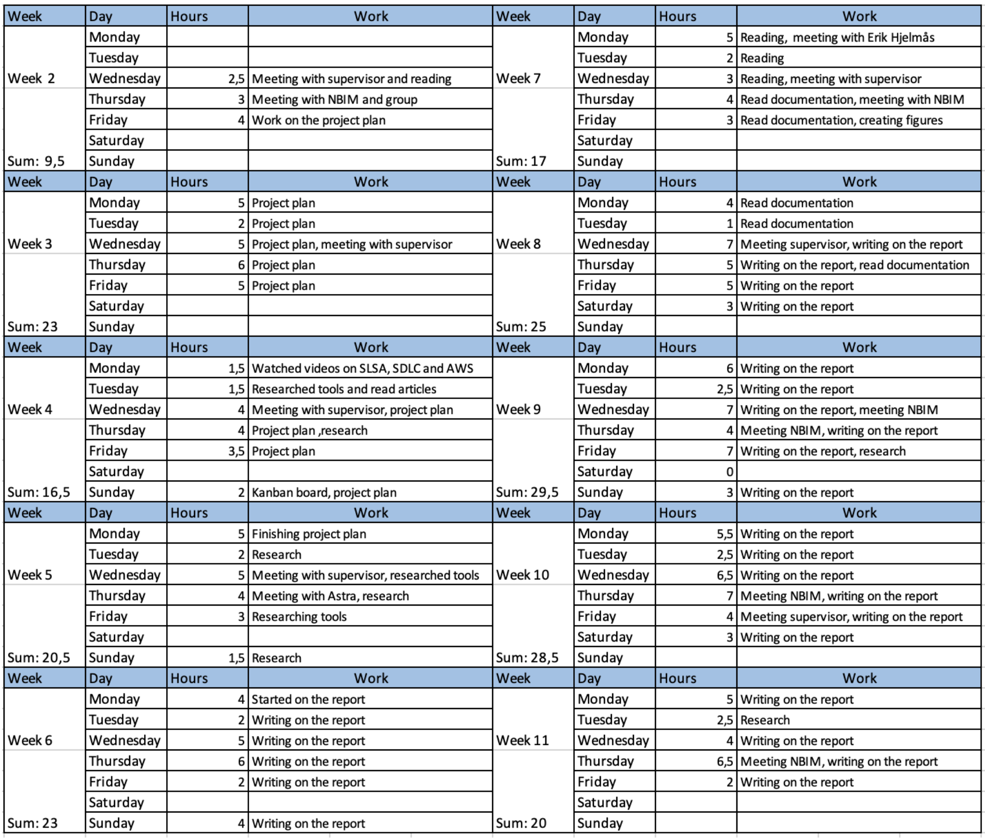
Thea Urne

Appendix C

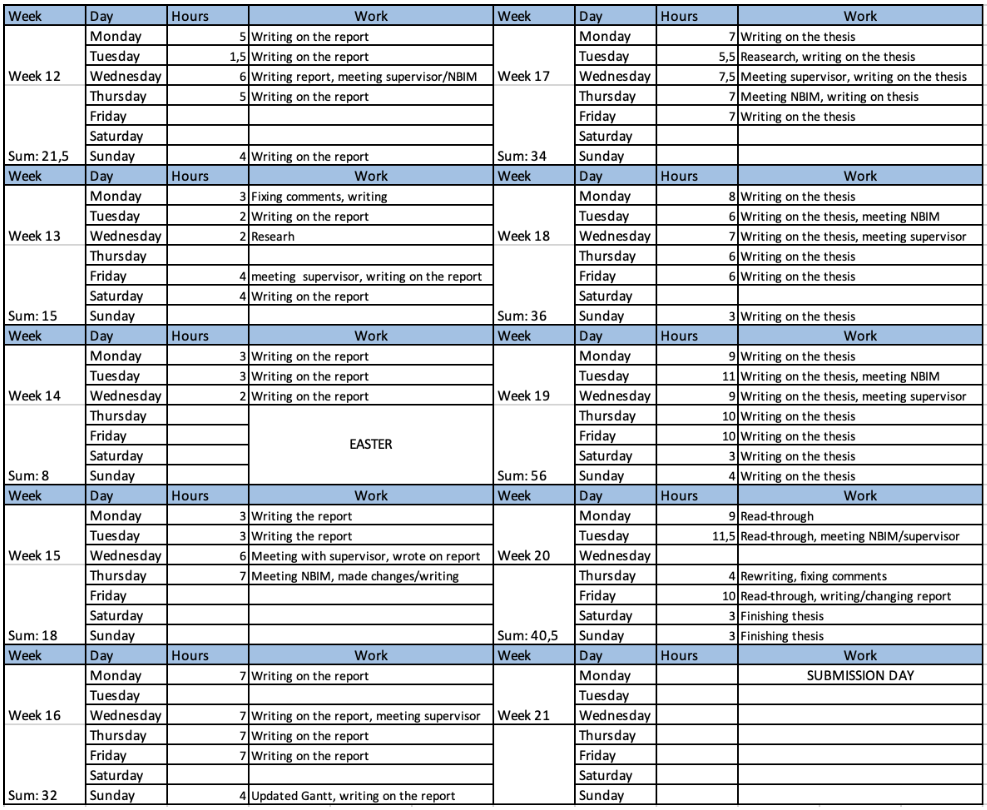
Timetables

The following are the individual timetables for the group members.

C.1 Timetable - Anniken

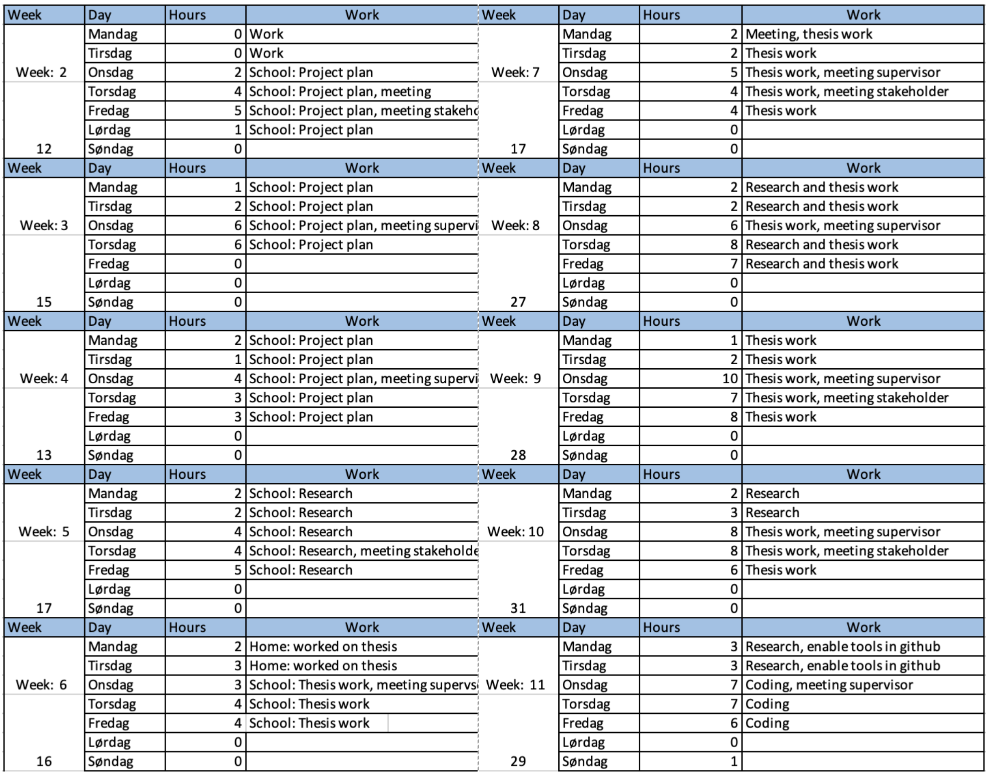


Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday				Monday	5	Reading, meeting with Erik Hjelmås
	Tuesday				Tuesday	2	Reading
Week 2	Wednesday	2,5	Meeting with supervisor and reading	Week 7	Wednesday	3	Reading, meeting with supervisor
	Thursday	3	Meeting with NBIM and group		Thursday	4	Read documentation, meeting with NBIM
	Friday	4	Work on the project plan		Friday	3	Read documentation, creating figures
	Saturday				Saturday		
Sum: 9,5	Sunday			Sum: 17	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	5	Project plan		Monday	4	Read documentation
	Tuesday	2	Project plan		Tuesday	1	Read documentation
Week 3	Wednesday	5	Project plan, meeting with supervisor	Week 8	Wednesday	7	Meeting supervisor, writing on the report
	Thursday	6	Project plan		Thursday	5	Writing on the report, read documentation
	Friday	5	Project plan		Friday	5	Writing on the report
	Saturday				Saturday	3	Writing on the report
Sum: 23	Sunday			Sum: 25	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	1,5	Watched videos on SLSA, SDLC and AWS		Monday	6	Writing on the report
	Tuesday	1,5	Researched tools and read articles		Tuesday	2,5	Writing on the report
Week 4	Wednesday	4	Meeting with supervisor, project plan	Week 9	Wednesday	7	Writing on the report, meeting NBIM
	Thursday	4	Project plan ,research		Thursday	4	Meeting NBIM, writing on the report
	Friday	3,5	Project plan		Friday	7	Writing on the report, research
	Saturday				Saturday	0	
Sum: 16,5	Sunday	2	Kanban board, project plan	Sum: 29,5	Sunday	3	Writing on the report
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	5	Finishing project plan		Monday	5,5	Writing on the report
	Tuesday	2	Research		Tuesday	2,5	Writing on the report
Week 5	Wednesday	5	Meeting with supervisor, researched tools	Week 10	Wednesday	6,5	Writing on the report
	Thursday	4	Meeting with Astra, research		Thursday	7	Meeting NBIM, writing on the report
	Friday	3	Researching tools		Friday	4	Meeting supervisor, writing on the report
	Saturday				Saturday	3	Writing on the report
Sum: 20,5	Sunday	1,5	Research	Sum: 28,5	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	4	Started on the report		Monday	5	Writing on the report
	Tuesday	2	Writing on the report		Tuesday	2,5	Research
Week 6	Wednesday	5	Writing on the report	Week 11	Wednesday	4	Writing on the report
	Thursday	6	Writing on the report		Thursday	6,5	Meeting NBIM, writing on the report
	Friday	2	Writing on the report		Friday	2	Writing on the report
	Saturday				Saturday		
Sum: 23	Sunday	4	Writing on the report	Sum: 20	Sunday		

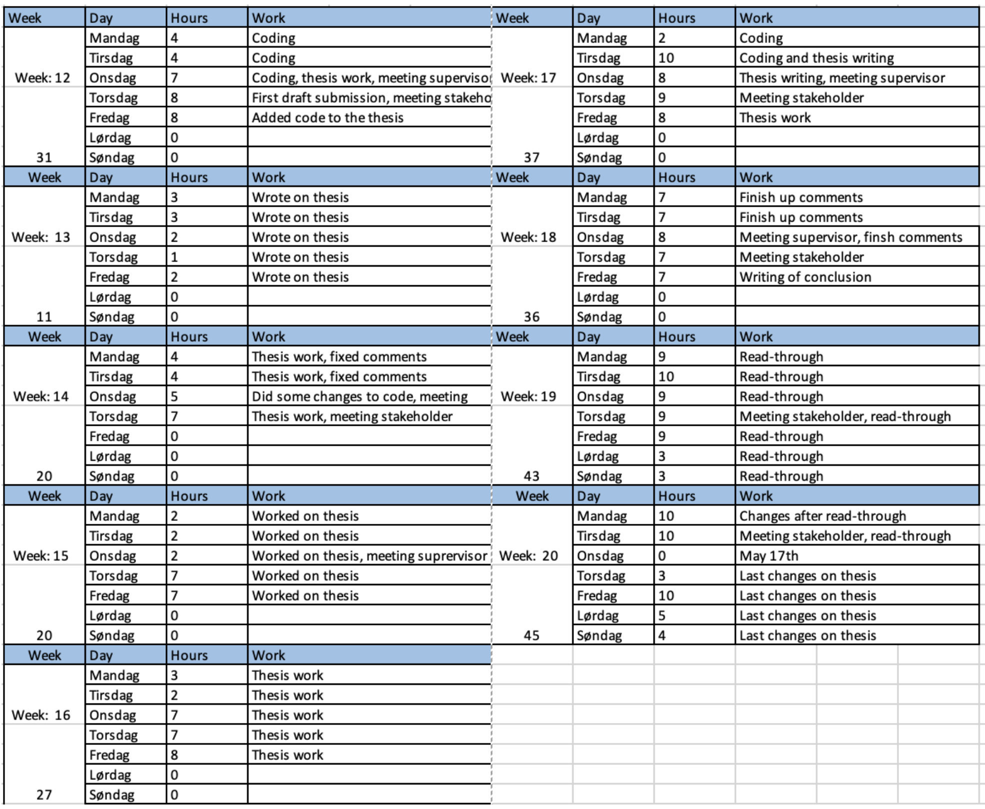


Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	5	Writing on the report		Monday	7	Writing on the thesis
	Tuesday	1,5	Writing on the report		Tuesday	5,5	Reasearch, writing on the thesis
Week 12	Wednesday	6	Writing report, meeting supervisor/NBIM	Week 17	Wednesday	7,5	Meeting supervisor, writing on the thesis
	Thursday	5	Writing on the report		Thursday	7	Meeting NBIM, writing on thesis
	Friday				Friday	7	Writing on the thesis
	Saturday				Saturday		
Sum: 21,5	Sunday	4	Writing on the report	Sum: 34	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	3	Fixing comments, writing		Monday	8	Writing on the thesis
	Tuesday	2	Writing on the report		Tuesday	6	Writing on the thesis, meeting NBIM
Week 13	Wednesday	2	Researh	Week 18	Wednesday	7	Writing on the thesis, meeting supervisor
	Thursday				Thursday	6	Writing on the thesis
	Friday	4	meeting supervisor, writing on the report		Friday	6	Writing on the thesis
	Saturday	4	Writing on the report		Saturday		
Sum: 15	Sunday			Sum: 36	Sunday	3	Writing on the thesis
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	3	Writing on the report		Monday	9	Writing on the thesis
	Tuesday	3	Writing on the report		Tuesday	11	Writing on the thesis, meeting NBIM
Week 14	Wednesday	2	Writing on the report	Week 19	Wednesday	9	Writing on the thesis, meeting supervisor
	Thursday		EASTER		Thursday	10	Writing on the thesis
	Friday				Friday	10	Writing on the thesis
	Saturday				Saturday	3	Writing on the thesis
Sum: 8	Sunday			Sum: 56	Sunday	4	Writing on the thesis
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	3	Writing the report		Monday	9	Read-through
	Tuesday	3	Writing the report		Tuesday	11,5	Read-through, meeting NBIM/supervisor
Week 15	Wednesday	6	Meeting with supervisor, wrote on report	Week 20	Wednesday		
	Thursday	7	Meeting NBIM, made changes/writing		Thursday	4	Rewriting, fixing comments
	Friday				Friday	10	Read-through, writing/changing report
	Saturday				Saturday	3	Finishing thesis
Sum: 18	Sunday			Sum: 40,5	Sunday	3	Finishing thesis
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	7	Writing on the report		Monday		SUBMISSION DAY
	Tuesday				Tuesday		
Week 16	Wednesday	7	Writing on the report, meeting supervisor	Week 21	Wednesday		
	Thursday	7	Writing on the report		Thursday		
	Friday	7	Writing on the report		Friday		
	Saturday				Saturday		
Sum: 32	Sunday	4	Updated Gantt, writing on the report		Sunday		

C.2 Timetable - Celina

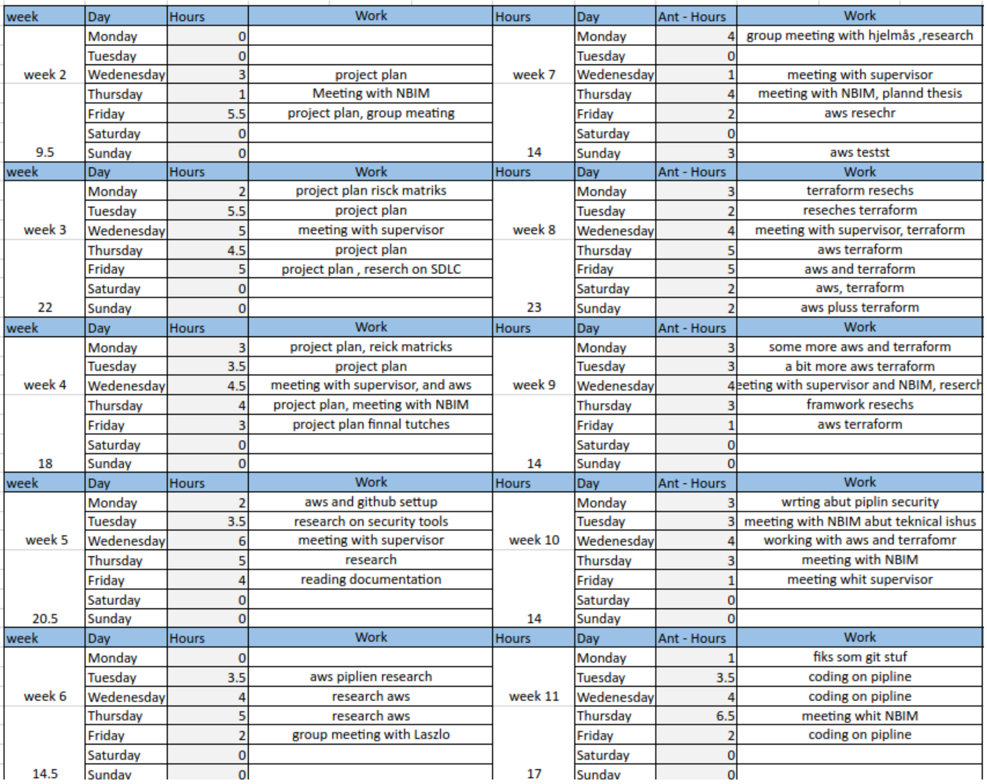


Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	0	Work		Mandag	2	Meeting, thesis work
	Tirsdag	0	Work		Tirsdag	2	Thesis work
Week: 2	Onsdag	2	School: Project plan	Week: 7	Onsdag	5	Thesis work, meeting supervisor
	Torsdag	4	School: Project plan, meeting		Torsdag	4	Thesis work, meeting stakeholder
	Fredag	5	School: Project plan, meeting stakehc		Fredag	4	Thesis work
	Lørdag	1	School: Project plan		Lørdag	0	
12	Søndag	0		17	Søndag	0	
Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	1	School: Project plan		Mandag	2	Research and thesis work
	Tirsdag	2	School: Project plan		Tirsdag	2	Research and thesis work
Week: 3	Onsdag	6	School: Project plan, meeting supervi	Week: 8	Onsdag	6	Thesis work, meeting supervisor
	Torsdag	6	School: Project plan		Torsdag	8	Research and thesis work
	Fredag	0			Fredag	7	Research and thesis work
	Lørdag	0			Lørdag	0	
15	Søndag	0		27	Søndag	0	
Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	2	School: Project plan		Mandag	1	Thesis work
	Tirsdag	1	School: Project plan		Tirsdag	2	Thesis work
Week: 4	Onsdag	4	School: Project plan, meeting supervi	Week: 9	Onsdag	10	Thesis work, meeting supervisor
	Torsdag	3	School: Project plan		Torsdag	7	Thesis work, meeting stakeholder
	Fredag	3	School: Project plan		Fredag	8	Thesis work
	Lørdag	0			Lørdag	0	
13	Søndag	0		28	Søndag	0	
Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	2	School: Research		Mandag	2	Research
	Tirsdag	2	School: Research		Tirsdag	3	Research
Week: 5	Onsdag	4	School: Research	Week: 10	Onsdag	8	Thesis work, meeting supervisor
	Torsdag	4	School: Research, meeting stakeholde		Torsdag	8	Thesis work, meeting stakeholder
	Fredag	5	School: Research		Fredag	6	Thesis work
	Lørdag	0			Lørdag	0	
17	Søndag	0		31	Søndag	0	
Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	2	Home: worked on thesis		Mandag	3	Research, enable tools in github
	Tirsdag	3	Home: worked on thesis		Tirsdag	3	Research, enable tools in github
Week: 6	Onsdag	3	School: Thesis work, meeting supervs	Week: 11	Onsdag	7	Coding, meeting supervisor
	Torsdag	4	School: Thesis work		Torsdag	7	Coding
	Fredag	4	School: Thesis work		Fredag	6	Coding
	Lørdag	0			Lørdag	0	
16	Søndag	0		29	Søndag	1	

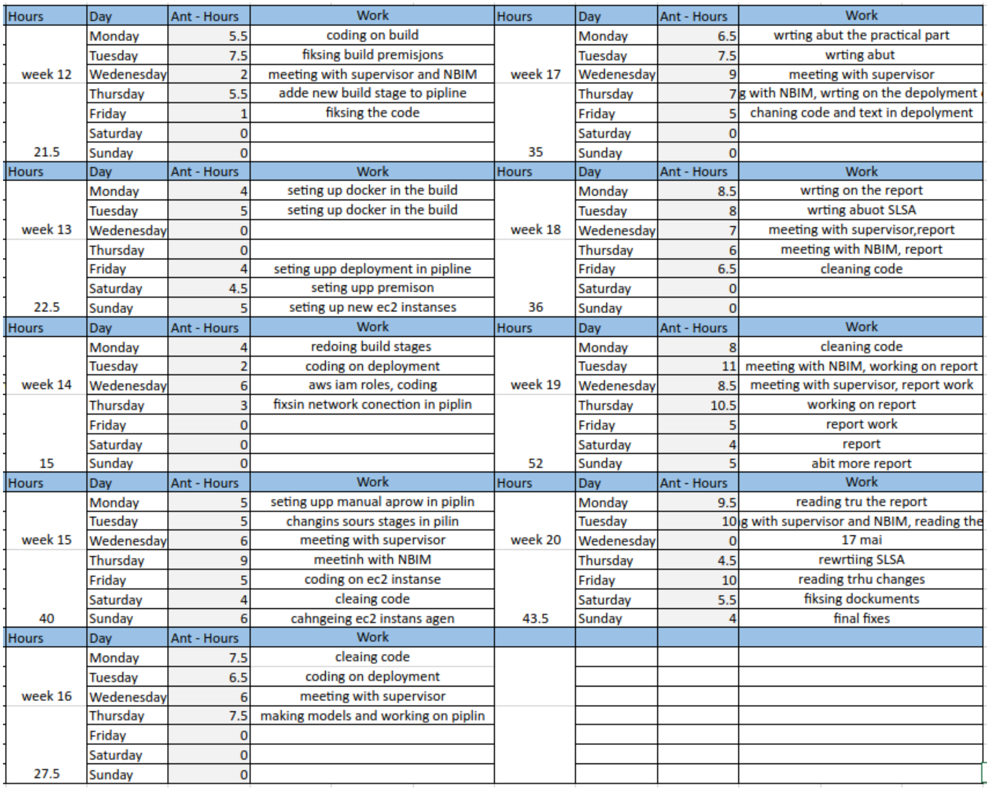


Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	4	Coding		Mandag	2	Coding
	Tirsdag	4	Coding		Tirsdag	10	Coding and thesis writing
Week: 12	Onsdag	7	Coding, thesis work, meeting supervisor	Week: 17	Onsdag	8	Thesis writing, meeting supervisor
	Torsdag	8	First draft submission, meeting stakeho		Torsdag	9	Meeting stakeholder
	Fredag	8	Added code to the thesis		Fredag	8	Thesis work
	Lørdag	0			Lørdag	0	
31	Søndag	0		37	Søndag	0	
Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	3	Wrote on thesis		Mandag	7	Finish up comments
	Tirsdag	3	Wrote on thesis		Tirsdag	7	Finish up comments
Week: 13	Onsdag	2	Wrote on thesis	Week: 18	Onsdag	8	Meeting supervisor, finsh comments
	Torsdag	1	Wrote on thesis		Torsdag	7	Meeting stakeholder
	Fredag	2	Wrote on thesis		Fredag	7	Writing of conclusion
	Lørdag	0			Lørdag	0	
11	Søndag	0		36	Søndag	0	
Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	4	Thesis work, fixed comments		Mandag	9	Read-through
	Tirsdag	4	Thesis work, fixed comments		Tirsdag	10	Read-through
Week: 14	Onsdag	5	Did some changes to code, meeting	Week: 19	Onsdag	9	Read-through
	Torsdag	7	Thesis work, meeting stakeholder		Torsdag	9	Meeting stakeholder, read-through
	Fredag	0			Fredag	9	Read-through
	Lørdag	0			Lørdag	3	Read-through
20	Søndag	0		43	Søndag	3	Read-through
Week	Day	Hours	Work	Week	Day	Hours	Work
	Mandag	2	Worked on thesis		Mandag	10	Changes after read-through
	Tirsdag	2	Worked on thesis		Tirsdag	10	Meeting stakeholder, read-through
Week: 15	Onsdag	2	Worked on thesis, meeting suprervisor	Week: 20	Onsdag	0	May 17th
	Torsdag	7	Worked on thesis		Torsdag	3	Last changes on thesis
	Fredag	7	Worked on thesis		Fredag	10	Last changes on thesis
	Lørdag	0			Lørdag	5	Last changes on thesis
20	Søndag	0		45	Søndag	4	Last changes on thesis
Week	Day	Hours	Work				
	Mandag	3	Thesis work				
	Tirsdag	2	Thesis work				
Week: 16	Onsdag	7	Thesis work				
	Torsdag	7	Thesis work				
	Fredag	8	Thesis work				
	Lørdag	0					
27	Søndag	0					

C.3 Timetable - Sebastian

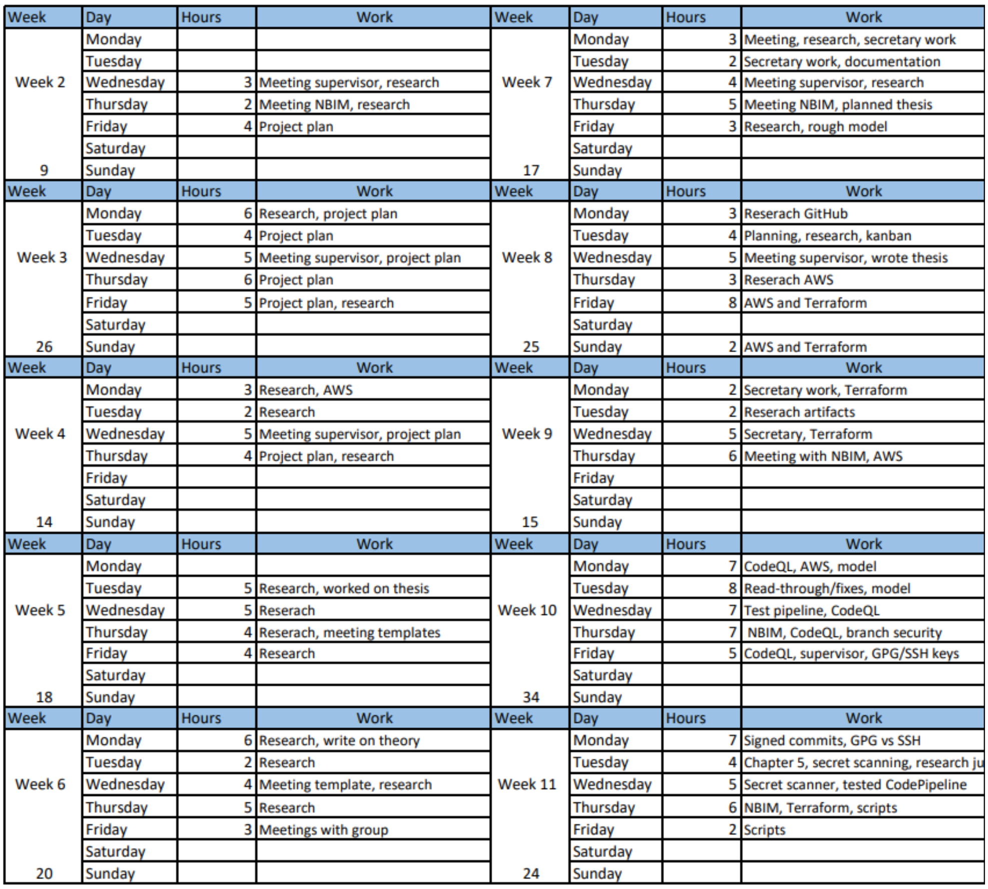


week	Day	Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	0			Monday	4	group meeting with hjelmås ,research
	Tuesday	0			Tuesday	0	
week 2	Wedenesday	3	project plan	week 7	Wedenesday	1	meeting with supervisor
	Thursday	1	Meeting with NBIM		Thursday	4	meeting with NBIM, plannd thesis
	Friday	5.5	project plan, group meating		Friday	2	aws resechr
	Saturday	0			Saturday	0	
9.5	Sunday	0		14	Sunday	3	aws testst
week	Day	Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	2	project plan risck matriks		Monday	3	terraform resechs
	Tuesday	5.5	project plan		Tuesday	2	reseches terraform
week 3	Wedenesday	5	meeting with supervisor	week 8	Wedenesday	4	meeting with supervisor, terraform
	Thursday	4.5	project plan		Thursday	5	aws terraform
	Friday	5	project plan , reserch on SDLC		Friday	5	aws and terraform
	Saturday	0			Saturday	2	aws, terraform
22	Sunday	0		23	Sunday	2	aws pluss terraform
week	Day	Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	3	project plan, reick matricks		Monday	3	some more aws and terraform
	Tuesday	3.5	project plan		Tuesday	3	a bit more aws terraform
week 4	Wedenesday	4.5	meeting with supervisor, and aws	week 9	Wedenesday	4	eeting with supervisor and NBIM, reserch
	Thursday	4	project plan, meeting with NBIM		Thursday	3	framwork resechs
	Friday	3	project plan finnal tutches		Friday	1	aws terraform
	Saturday	0			Saturday	0	
18	Sunday	0		14	Sunday	0	
week	Day	Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	2	aws and github settup		Monday	3	wrting abut piplin security
	Tuesday	3.5	research on security tools		Tuesday	3	meeting with NBIM abut teknical ishus
week 5	Wedenesday	6	meeting with supervisor	week 10	Wedenesday	4	working with aws and terrafomr
	Thursday	5	research		Thursday	3	meeting with NBIM
	Friday	4	reading documentation		Friday	1	meeting whit supervisor
	Saturday	0			Saturday	0	
20.5	Sunday	0		14	Sunday	0	
week	Day	Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	0			Monday	1	fiks som git stuf
	Tuesday	3.5	aws piplien research		Tuesday	3.5	coding on pipline
week 6	Wedenesday	4	research aws	week 11	Wedenesday	4	coding on pipline
	Thursday	5	research aws		Thursday	6.5	meeting whit NBIM
	Friday	2	group meeting with Laszlo		Friday	2	coding on pipline
	Saturday	0			Saturday	0	
14.5	Sunday	0		17	Sunday	0	

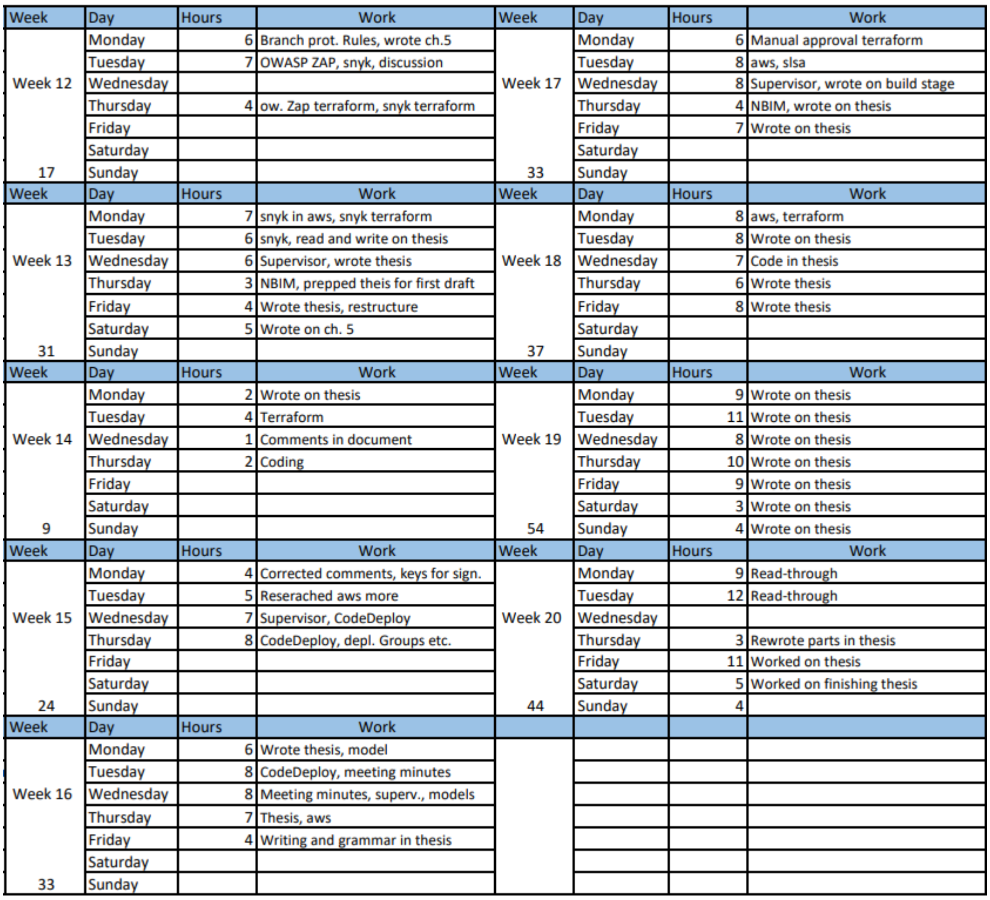


Hours	Day	Ant - Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	5.5	coding on build		Monday	6.5	wrting abut the practical part
	Tuesday	7.5	fiksing build premisjons		Tuesday	7.5	wrting abut
week 12	Wedenesday	2	meeting with supervisor and NBIM	week 17	Wedenesday	9	meeting with supervisor
	Thursday	5.5	adde new build stage to pipline		Thursday	7	g with NBIM, wrting on the depolyment
	Friday	1	fiksing the code		Friday	5	chaning code and text in depolyment
	Saturday	0			Saturday	0	
21.5	Sunday	0		35	Sunday	0	
Hours	Day	Ant - Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	4	seting up docker in the build		Monday	8.5	wrting on the report
	Tuesday	5	seting up docker in the build		Tuesday	8	wrting abuot SLSA
week 13	Wedenesday	0		week 18	Wedenesday	7	meeting with supervisor,report
	Thursday	0			Thursday	6	meeting with NBIM, report
	Friday	4	seting upp deployment in pipline		Friday	6.5	cleaning code
	Saturday	4.5	seting upp premison		Saturday	0	
22.5	Sunday	5	seting up new ec2 instanses	36	Sunday	0	
Hours	Day	Ant - Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	4	redoing build stages		Monday	8	cleaning code
	Tuesday	2	coding on deployment		Tuesday	11	meeting with NBIM, working on report
week 14	Wedenesday	6	aws iam roles, coding	week 19	Wedenesday	8.5	meeting with supervisor, report work
	Thursday	3	fixsin network conection in piplin		Thursday	10.5	working on report
	Friday	0			Friday	5	report work
	Saturday	0			Saturday	4	report
15	Sunday	0		52	Sunday	5	abit more report
Hours	Day	Ant - Hours	Work	Hours	Day	Ant - Hours	Work
	Monday	5	seting upp manual aprow in piplin		Monday	9.5	reading tru the report
	Tuesday	5	changins sours stages in pilin		Tuesday	10	g with supervisor and NBIM, reading the
week 15	Wedenesday	6	meeting with supervisor	week 20	Wedenesday	0	17 mai
	Thursday	9	meetinh with NBIM		Thursday	4.5	rewrtiing SLSA
	Friday	5	coding on ec2 instanse		Friday	10	reading trhu changes
	Saturday	4	cleaing code		Saturday	5.5	fiksing dockuments
40	Sunday	6	cahngeing ec2 instans agen	43.5	Sunday	4	final fixes
Hours	Day	Ant - Hours	Work				
	Monday	7.5	cleaing code				
	Tuesday	6.5	coding on deployment				
week 16	Wedenesday	6	meeting with supervisor				
	Thursday	7.5	making models and working on piplin				
	Friday	0					
	Saturday	0					
27.5	Sunday	0					

C.4 Timetable - Thea



Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday				Monday	3	Meeting, research, secretary work
	Tuesday				Tuesday	2	Secretary work, documentation
Week 2	Wednesday	3	Meeting supervisor, research	Week 7	Wednesday	4	Meeting supervisor, research
	Thursday	2	Meeting NBIM, research		Thursday	5	Meeting NBIM, planned thesis
	Friday	4	Project plan		Friday	3	Research, rough model
	Saturday				Saturday		
9	Sunday			17	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	6	Research, project plan		Monday	3	Reserach GitHub
	Tuesday	4	Project plan		Tuesday	4	Planning, research, kanban
Week 3	Wednesday	5	Meeting supervisor, project plan	Week 8	Wednesday	5	Meeting supervisor, wrote thesis
	Thursday	6	Project plan		Thursday	3	Reserach AWS
	Friday	5	Project plan, research		Friday	8	AWS and Terraform
	Saturday				Saturday		
26	Sunday			25	Sunday	2	AWS and Terraform
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	3	Research, AWS		Monday	2	Secretary work, Terraform
	Tuesday	2	Research		Tuesday	2	Reserach artifacts
Week 4	Wednesday	5	Meeting supervisor, project plan	Week 9	Wednesday	5	Secretary, Terraform
	Thursday	4	Project plan, research		Thursday	6	Meeting with NBIM, AWS
	Friday				Friday		
	Saturday				Saturday		
14	Sunday			15	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday				Monday	7	CodeQL, AWS, model
	Tuesday	5	Research, worked on thesis		Tuesday	8	Read-through/fixes, model
Week 5	Wednesday	5	Reserach	Week 10	Wednesday	7	Test pipeline, CodeQL
	Thursday	4	Reserach, meeting templates		Thursday	7	NBIM, CodeQL, branch security
	Friday	4	Research		Friday	5	CodeQL, supervisor, GPG/SSH keys
	Saturday				Saturday		
18	Sunday			34	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	6	Research, write on theory		Monday	7	Signed commits, GPG vs SSH
	Tuesday	2	Research		Tuesday	4	Chapter 5, secret scanning, research ju
Week 6	Wednesday	4	Meeting template, research	Week 11	Wednesday	5	Secret scanner, tested CodePipeline
	Thursday	5	Research		Thursday	6	NBIM, Terraform, scripts
	Friday	3	Meetings with group		Friday	2	Scripts
	Saturday				Saturday		
20	Sunday			24	Sunday		



Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	6	Branch prot. Rules, wrote ch.5		Monday	6	Manual approval terraform
	Tuesday	7	OWASP ZAP, snyk, discussion		Tuesday	8	aws, slsa
Week 12	Wednesday			Week 17	Wednesday	8	Supervisor, wrote on build stage
	Thursday	4	ow. Zap terraform, snyk terraform		Thursday	4	NBIM, wrote on thesis
	Friday				Friday	7	Wrote on thesis
	Saturday				Saturday		
17	Sunday			33	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	7	snyk in aws, snyk terraform		Monday	8	aws, terraform
	Tuesday	6	snyk, read and write on thesis		Tuesday	8	Wrote on thesis
Week 13	Wednesday	6	Supervisor, wrote thesis	Week 18	Wednesday	7	Code in thesis
	Thursday	3	NBIM, prepped theis for first draft		Thursday	6	Wrote thesis
	Friday	4	Wrote thesis, restructure		Friday	8	Wrote thesis
	Saturday	5	Wrote on ch. 5		Saturday		
31	Sunday			37	Sunday		
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	2	Wrote on thesis		Monday	9	Wrote on thesis
	Tuesday	4	Terraform		Tuesday	11	Wrote on thesis
Week 14	Wednesday	1	Comments in document	Week 19	Wednesday	8	Wrote on thesis
	Thursday	2	Coding		Thursday	10	Wrote on thesis
	Friday				Friday	9	Wrote on thesis
	Saturday				Saturday	3	Wrote on thesis
9	Sunday			54	Sunday	4	Wrote on thesis
Week	Day	Hours	Work	Week	Day	Hours	Work
	Monday	4	Corrected comments, keys for sign.		Monday	9	Read-through
	Tuesday	5	Reserached aws more		Tuesday	12	Read-through
Week 15	Wednesday	7	Supervisor, CodeDeploy	Week 20	Wednesday		
	Thursday	8	CodeDeploy, depl. Groups etc.		Thursday	3	Rewrote parts in thesis
	Friday				Friday	11	Worked on thesis
	Saturday				Saturday	5	Worked on finishing thesis
24	Sunday			44	Sunday	4	
Week	Day	Hours	Work				
	Monday	6	Wrote thesis, model				
	Tuesday	8	CodeDeploy, meeting minutes				
Week 16	Wednesday	8	Meeting minutes, superv., models				
	Thursday	7	Thesis, aws				
	Friday	4	Writing and grammar in thesis				
	Saturday						
33	Sunday						

Appendix D

Meeting minutes from meetings with the supervisor

D.1 25 January 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-11.00

Place: NTNU Campus Gjøvik, A123

Project plan

- Writing style: avoid including the reader
- Write about the group from one perspective
- To get better grade – put in extra work, for example practical work
- Add paragraph about Scrum

Other

- Waiting for NBIM to get the project agreement ready
  - Will be signing digitally

D.2 1 February 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-10.40

Place: NTNU Campus Gjøvik, A132

Scientific articles – resources

- IEEE explore
- Springer link
- MDPI

What should we do now?

- Research
- Get the base knowledge

Other

- Next weeks meeting is cancelled

D.3 15 February 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-11.00

Place: NTNU Campus Gjøvik, A132

Dates for delivering drafts

- 1st draft - 3 April
- Final draft - 1 May

Discussion about the thesis

- Shift of approach
  - More focus on securing the different steps of the pipeline

Other

- Group should look for templates for the thesis on blackboard
- Alternatively look at what other groups have included

D.4 22 February 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-10.50

Place: NTNU Campus Gjøvik, A132

Other

- Status check - group felt lost for a while, but it is better now
- Showed the supervisor our plan for the approach - approved
- Next meeting moved to 13.30 - after the meeting with NBIM

D.5 1 March 2023

Participants: Anniken, Celina, Filip and Sebastian

Time: 13.30-14.00

Place: NTNU Campus Gjøvik, A132

Other

- What the group learned from the crash course on writing a thesis
- Talked more to the stakeholder on what they want the group to do

Discussion about the thesis

- What type of chapters to include
- Read through parts of the thesis
- Talked on how the thesis should be structured
- We do not have to include all meeting minutes from the scrum meetings

D.6 10 March 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-11.10

Place: NTNU Campus Gjøvik, A155.4

Discussion about the thesis

- Group is doing well – have done a lot of work and is pleased with the result so far
- Looked through some of the comments made in the thesis to clear up some confusion
- Group wondered if it was ok to use blog posts as sources
    - That's ok if the source is reliable
- Discussed how to do citing in the thesis – group is used to "[..]", but we should look into that further
- Could look at including footnotes for websites to tools we write about, e.g. Snyk
- For the practical part of the project: group thought of making a use case of a pipeline with implemented security tools, and works on automation of pipelines with Terraform code
- When using CodeQL – show examples of critical, high, medium and low level alerts
- Could be beneficial to look at another SAST tool (either from GitHub or a third party)

Other

- Next meeting is cancelled

D.7 22 March 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-10.40

Place: NTNU Campus Gjøvik, A132

Discussion about the thesis

- Good idea to implement practical work - like how we have built a secure pipeline
- Things are a bit all over the place now - it will come together eventually
- We should describe the work pipeline
- Some sections are going to be removed - like Testing
- Should include a link to our GitHub - the most important parts should be included as an appendix
- We should write about maintenance security

D.8 29 March 2023

Participants: Celina, Filip and Thea

Time: 10.30-11.00

Place: NTNU Campus Gjøvik, A132

Overall

- We went through all comments made in the overleaf document - added the answers to the comments
- Delivering 1st draft 3 April - soon ready
- Cancel next week's meeting

D.9 12 April 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-10.40

Place: NTNU Campus Gjøvik, A132

- Went thorugh the coments in the document
- He read through first draft - it was mostly good, some confusion about the "Discussion" chapter

D.10 19 April 2023

Participants: Anniken, Filip, Sebastian and Thea

Time: 10.30-11.00

Place: NTNU Campus Gjøvik, A132

Discussion about the thesis

- We should include discussion on both the finds and our work in the same chapter, but separate it into sections
- In the "pipeline" chapter - remove snyk since we never used it, add a table for pros and cons to OWASP Zap.
- Have started to include meeting minutes (notes from meetings written in Teams) - taken inspiration from other theses

Other

- We are on time to be able to deliver final draft 1 May, if not we will deliver it at least the same week

D.11 26 April 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-11.00

Place: NTNU Campus Gjøvik, A132

Discussion about the thesis

- We have a working practical prototype - we will add it to the deployment chapter soon
- In the deployment chapter we will add models and explanations of what happens through the pipeline – make it understandable
- For the presentation – maybe record the process and speed it up to demonstrate
- Thinking of connecting our own work to the practices we recommend and SDLC etc
- Included frameworks – good idea, will read through and give us feedback on where it should be
- Implement frameworks to our own work
- Footnotes are ugly, can't fix it so it's fine
- Write a subsection on why we chose integrated GitHub tools and AWS services
- Pros and cons for AWS services – add it for the different services, based mostly on our own opinions, and try to find sources as well
-

Other

- 1st May is a holiday – will deliver final draft 2nd or 3rd May

D.12 3 May 2023

Participants: Anniken, Filip, Sebastian and Thea

Time: 10.30-11.00

Place: NTNU Campus Gjøvik, A132

Discussion about the thesis

- Went through some of the comments made in the document
- Not have “our” in the titles in Discussion
- Some rephrasing in Deployment
- Output from OWASP Zap - make into text
- When writing about meetings , reference the appendix
- For framework - write about SLSA and level we hit, what can be done for hitting higher levels
- What is left - add the rest in all chapters, summary and conclusion, fine-tuning the text

Other

- Delivered the final draft yesterday
- Filip will read through it on Sunday so if we want to deliver a newer version, this can be done by the end of the week

D.13 10 May 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-11.00

Place: NTNU Campus Gjøvik, A132

Discussion about the thesis

- Went through some of the comments Filip had made in the final draft
  - Change some of the checkmarks in the Table 2.5 to be something else - looks like it is positive, but it is not
  - Mention of AWS and GitHub in security of the pipeline, but it is not supposed to mention any tools? We mention it because AWS and GitHub are what is "required" of us to use and is presented as that earlier
  - Some of the footnotes need rephrasing
- When using a source for the whole paragraph, should we include it at the end of the first sentence or at the end of the last? Do whatever we want, just be consistent.
- What code should we include in the appendix? The main code could be in the appendix, smaller parts are in the repository. See what other theses have done before

Discussion about the thesis

- Meeting next week moved to Tuesday because of 17th of May

D.14 16 May 2023

Participants: Anniken, Celina, Filip, Sebastian and Thea

Time: 10.30-11.00

Place: NTNU Campus Gjøvik, A221

Discussion about the thesis

- New title for chapter 5 - suggest “building a secure end-to-end pipeline” since that is what we do
- We are removing the code in the appendix and will just link to the different repositories in a footnote
  - We have a GitHub organization with several repositories for different code purposes
- Not necessary to link to the different CWEs from the alerts - also, refer to CVE from the theory
- Dependabot refers to a ghsa id, refers to GitHub Advisory Database, refers further to the same CVE id. Should we include? No, not necessary
- Framework in discussion: have an introduction on what we want to do with it. Something in bold text in SLSA to be more consistent.
- Change “relevant hyperlink” to “code repository” or something
- Keywords in abstract in alphabetical order
- Rewrite some of the glossary entries, or add sources if it is directly taken from somewhere
- Code when delivering – should we deliver the whole juice shop? We have made some small changes.
  - Only include the modified files, specify that it is the appspec file is our code and needs to be added to the juice-shop. Explain in the README file.

Appendix E

Meeting minutes from meetings with stakeholder

E.1 12 January 2023

Participants: Anniken, Astri, Celina, Sebastian, Stian, Stian and Thea

Time: 12.00-12.30

Place: Teams

- Went through standard agreement
  - Celina will send them the agreement and confidentiality agreement from NTNU
  - NBIM will go through these and make necessary changes
- In our thesis, we will use GitHub, AWS and other public systems
- As a start, the group will get to know SDLC and AWS
- Set up an example of workflow, from beginning to end
- Group and NBIM will meet at the same time every other week to begin with
  - Will see if more is necessary later

E.2 26 January 2023

Participants: Anniken, Celina, Sebastian, Stian and Stian

Time: 12.00-12.30

Place: Teams

Other

- General talk about the contract and how we are going to sign it.
- How they will cover expense for the project relating to software we are going to use.

Discussion about the thesis

- We are only going to look at point four to six of SDLC
- Mabey think about signatures and artefacts in the code for validate the integrity.
- Only test with one code language
- Look into how many warnings that we need to put out in the pipeline (what is user friendly)
- Have general tests.

E.3 16 February 2023

Participants: Anniken, Celina, Sebastian, Stian, Stian and Thea

Time: 12.00-12.30

Place: Teams

Discussion about the thesis

- Might want to include false positives, artifacts, signatures etc.

For next week

- Make a pipeline and some examples of how we want to do it for a better overview

Other

- Having some issues with AWS - will try to fix ourselves, if not we will contact them to try to help us

E.4 1 March 2023

Participants: Anniken, Celina, Sebastian, Stian, Stian and Thea

Time: 12.00-13.00

Place: Teams

Discussion about the thesis

- Build upon earlier bachelor theses
- Scrap IAST and RAST
- We can get SAST and SCA scans from GitHub
- Look at GitHub Actions
- One cannot set up checks for signes code in AWS, but one can set up a point in the pipeline
- Look into juice-shop (git repo)
- Dependabot does not support all languages, but should not be any less good than others
- Look at other papers about tools and base it on this. There is a lot of similar papers out there

Practical

- Get an end to end as fast as possible (Secret Scanning, Dependabot)
- Add scanning tools from AWS (container scanning)
- Implement CodeQL and SCA when we get the pipeline to work
- Nice to use what GitHub already offers to get GitHub Actions to run
- Set branch protection – look into this
- GPG keys in GitHub – look into this

Other

- We will send them updated sprint plan
- Set up weekly meetings

E.5 9 March 2023

Participants: Anniken, Astri, Celina, Sebastian, Stian, Stian and Thea

Time: 12.00-12.30

Place: Teams

Discussion about the thesis

- Look more into DAST tools – so far, we have no specific
- DDoS protection
- We have looked more at signatures – AWS artifact signing
  - KMS cryptography
  - Need to configure ourselves (?)
- Lambda – supports signings in bundles, not docker image
- Compare GPG vs SSH keys for signing - Security vs user friendliness
- Find a paper with test case of eg juice-shop to compare tools
- Start with test case soon – try the whole pipeline
- Look more into CodeQL in VScode
- Could look at push protection – might not be so user friendly?
- Can use their colour palette but not their logo

E.6 16 March 2023

Participants: Anniken, Celina, Sebastian, Stian, Stian and Thea

Time: 12.00-12.30

Place: Teams

Practical work in the thesis

- Write about the complexity of the work rather then a step-by-step user guide
- Maybe include terraform code for security in Git

Analysis of tools

- Point to other's analysis of tools rather then doing it ourselves
- Might only end up with maximum of two tools each step - so might not be that important to include
- Possible to discuss in the discussion chapter

Basic security

- Can briefly include basic security - such as access control and MFA

Other

- Stian will look at GPG vs SSH keys - we read GPG was better, but he had heard otherwise

E.7 22 March 2023

Participants: Anniken, Celina, Sebastian, Stian, Stian and Thea

Time: 12.00-12.20

Place: Teams

GPG vs SSH

- No difference in the two in signing, more on key management, revoke, expire time
- SSH own file for allowed and revoked keys
- We should consider if it is worth even comparing

E.8 30 March 2023

Participants: Celina, Stian, Stian and Thea

Time: 12.00-12.30

Place: Teams

Discussion about the thesis

- Unsure how to implement code in the thesis - talk to supervisor about it
- Code as appendix - write about it without necessarily showing it
- Consider if we should build the text with focus on first time users or for more advanced users.
- One angle might be if there is too much implementation, or some is more difficult to integrate, we could make a table where we rank based on user friendliness.
- Could look at the amount of notifications you receive - not so good if you get spammed with notifications
- Talking about security in deployment - separate between deployment to testing and deployment to production
- Security after deployment - becomes maintenance (maintain the security). Can use different tools in AWS, like protect incoming traffic, monitoring etc. This is a lot of work, but can be mentioned
- Important to check what we want to deploy is what has been deployed

E.9 13 April 2023

Participants: Anniken, Celina, Sebastian, Stian, Stian and Thea

Time: 12.00-12.30

Place: Teams

Discussion about the thesis

- Practical part is on track, expected to be done very soon
- Looking at different ways of targeting the pipeline
  - Loop
  - Applications
- More clear definition on what steps of the pipeline and security measures for the different phases
- Back up the table of comparison of DAST, SAST and SCA with concrete examples why we put it there.

E.10 21 April 2023

Participants: Anniken, Celina, Stian, Stian and Thea

Time: 12.00-12.40

Place: Teams

Discussion about the thesis

- Find a way to sign the artifacts in S3 buckets - AWS CodeArtifact (artifact management)
- Figure out where to sign - need to look closer into what CodeDeploy actually does
- Trying to automate CodeQL, Dependabot, and secret scanner
- Would be nice to automate everything (end-to-end)
- On track for the final draft to be finished - we are focusing on getting as much text in as possible for the delivery
- They think the structure looks good so far
- Figures look better and more explanatory than before
- We are going to add the steps from the pipeline into the phases of the SDLC
- Sent them out GitHub repository for the Terraform code
- Could be better to use Lambda instead of EC2 - less complicated
- Nice with a screenshot of security alerts and a little explanation
- Could look into how many alerts for CodeQL vs the amount of known vulnerabilities - check juice-shop and other code

E.11 27 April 2023

Participants: Anniken, Celina, Sebastian, Stian, and Stian

Time: 12.15-12.45

Place: Teams

Discussion about the thesis

- The practical part looks good (add in a future step that you don't use*)
- Check options for signed artifacts
- Use the SHA-256 file with GPG and check if the hash is correct
- We can discuss the framework and how we meet/fail to meet the requirements for the various levels

Other

- We will try to deliver the final draft on the 1st of May, and the stakeholder will receive a PDF version of the draft so they can read through it

E.12 4 May 2023

Participants: Anniken, Celina, Sebastian, Stian, Stian and Thea

Time: 12.00-12.15

Place: Teams

De skal lese gjennom hele rapporten før helga Kan sette opp flere møter nå som det nærmer seg

Discussion about the thesis

- Is it necessary to include the work process?
  - Have seen from previous groups that it is normal to discuss the process
- Should get the overall picture to come better together
  - Why things fit together
  - What is important to focus on
- Should highlight the most important points at the end of the paragraphs
- For the figures - should be able to understand the figure without having read the text
- Other than this, the report is coming together well
- We are struggling a bit with the deployment chapter
  - Don't know how advanced it should be - the level of knowledge the reader has
  - Have gotten conflicting messages from the supervisor from NTNU and NBIM - one says it should be detailed, but the other says it should be a brief explanation
- Should consider excluding open source as a pro/con - we say it is a bad thing, but Stian has a more positive view on open source. Maybe better to not mention it?

Other

- They will try to read through the whole report before the weekend

- We can set up more meetings if needed now that the deadline is approaching

E.13 9 May 2023

Participants: Anniken, Astri, Celina, Sebastian, Stian, Stian and Thea

Time: 14.00-14.50

Place: Teams

Discussion about the thesis - Structure

- The thesis is very fragmented - maybe add some discussion before?
- Big change - we followed standard structure
- Example: which tool we chose, how to implement it, what we thought of it, all in one chapter?
- Out structure so far - introduce the theory, then present pipeline on a general basis without any specific tools, then the tools we use, and then how to implement it. Can read one without the other
- Talk about it with supervisor tomorrow

Discussion about the thesis - Framework

- Very detailed (reference to sections) - more curious on the overall assessment.
- More highlevel discussion about if it is useful - consider work vs. utility
- Would we use it if we could choose? Or is it just more work?

Discussion about the thesis - Conclusion

- Other theses have had personal and theoretical conclusion combined, that is why we have it. Maybe a bit messy?
- Theoretical conclusion should maybe come after the theoretical discussion?

Discussion about the thesis - General

- Should discuss where the practical part ended compared to the theoretical part
- Signing in Git but is not verified in AWS?
- Verification should happen at every stage in the pipeline - not necessary where everything runs in the same stage

- Access control in the pipeline model - will maybe look messy to include one box for every stage. Maybe have a “global box” that refer to acccess control at every stage
- SAST and DAST in the pipeline model - only suggestions on where to have it. We only include it in one of the suggested stages
- SDLC phases - include that everything can be repeated. Takes time to set up first time, but is automated when repeated
- Deployment in the SDLC - more focused on automation, first time setup, reuse
- Scope limitation - perceived as apologetic
- Maybe look into SAST and DAST in consideration to AI?
- Astri did not experience SAST as easy to set up. Requires a lot of tweaking, get a lot of warnings
- We will set up Grammarly for help with academic writing
- Have sharpened the policies for the S3 bucket - no longer has access to everything
- Try to simplify the deploymetn chapter - only write about the necessary and important things, maybe include more code than text for some parts? Hard because there are so many components to set up for everything to work.
- Use footnotes more consistently

E.14 16 May 2023

Participants: Anniken, Celina, Sebastian, Stian, Stian and Thea

Time: 14.00-14.30

Place: Teams

Discussion about the thesis

- In the discussion, we have written that we are not making software, but the process is same as SDLC - Distinction of iac and application code, not necessary. The process of writing and developing, make it clear.
- Phrase idempotent – professor thinks it is ok, they are not used to it, but since it is correct then its fine. Used in the context of iac
- Table comparison – supervisor told us to change symbol for negative checks. We are leaving it as is. Specify "many false positives" for examples.
- Read write permissions for CodePipeline – deploy only need read access. Can look at sharpening these permissions. Only add a comment, not spend too much time on it
- The point of SDLC being iterative is fine
- Chapter 5 is better and easier to understand now
- Usefulness of the framework – added more feedback to it
- Most feedback now are small details
- We need to fix abstract – need to make a good first impression
- Some details in the project plan - duplicate section and some small comments
- Why some code in appendix and other not? We have decided to have all code in GitHub, none in appendix.
- Change some of the captions of the figured – more explanatory

Other

- If we need feedback – contact them directly, don't need next meeting since delivery is on Monday

- If they have time, maybe give us feedback in writing on what they think of our work, their view of the result and the process
- We can also give them feedback on what we think of the work with them - after delivery

Appendix F

Meeting minutes from meetings with others

Meeting minutes from meetings the group has had with other participants. These include:

- Erik Hjelmås
- Laszlo Erdodi

F.1 6 February 2023

Participants: Anniken, Celina, Laszlo Erdodi, Sebastian and Thea

Time: 11.30-12.00

Place: Teams

Discussion on how to test code

- We should look at high level code and low-level code, because they have different type of security flaws we can test
- We got recommended to look at different vulnerabilities databases and look at the code that is attached to some of the vulnerabilities
- Not recommended to test PHP code
- We should use longer code bits because that is a more realistic test
- Useful links:
  - https://joern.io/
  - https://www.cvedetails.com/browse-by-date.php

F.2 13 February 2023

Participants: Anniken, Erik Hjelmås, Sebastian and Thea

Time: 09.30-10.00

Place: NTNU Campus Gjøvik, T540

Discussion about what to do for the thesis

- Group is feeling a little lost what to do and where to go on with the report - hoping Erik had some input on what to include and what to focus on
- The group should ask the stakeholder what exactly they imagine getting tested
- Should maybe test large open source projects
- Should look into GitHub and AWS tools
- For the thesis, a suggestion is to go in depth on the tools GitHub and AWS offers, make a secure pipeline, rather than comparing tools - these tools are already state of the art
- Should start by making some models showing the flow of development, visualizing all steps
- Start by coding $->$ docker containers $->$ etc.
- Should look into usinex SDLC Security report, and other reports from usinex (quality assured)
- Look into what Microsoft says about best practice in securing the SDLC

Appendix G

Standard agreement

Standard agreement between the group, NTNU and NBIM.

NTNU

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for informasjonssikkerhet og kommunikasjonsteknologi
Veileder ved NTNU: Filip Holik e-post og tlf. Filip.holik@ntnu.no, tlf. +420 730 640 580
Ekstern virksomhet: Norges Bank - avdeling Norges Bank Investment Management Ekstern virksomhet sin kontaktperson, e-post og tlf.: Stian Hagbø Olsen, stian.hagbo.olsen@nbim.no, tlf. 2407 3255 Stian Kristoffersen, stian.kristoffersen@nbim.no, tlf. 2407 3865
Student: Anniken Arildset Fødselsdato: 07.04.00
Student: Celina Heimdal Brynildsen Fødselsdato: 15.07.01
Student: Sebastian Hestsveen Fødselsdato: 25.12.00
Student: Thea Urne Fødselsdato: 08.01.01

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: 09.01.23
Sluttdato: 22.05.23

Oppgavens arbeidstittel er: Securing the Software Development Life Cycle

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:
NBIM dekker kostnader til innkjøp av software lisenser og liknende, oppad begrenset til totalt kr. 8000 (ex. mva). Kostnadene refunderes mot fremleggelse av kvitteringer. Studentene må bli enige seg imellom om hvordan dette totalbeløpet skal benyttes.

For ordens skyld, NBIM dekker ikke kostnader til reiser.

Ekstern virksomhet stiller med to kontaktpersoner, som hver vil bidra med én time per uke for hele studentgruppen (ikke per student). Veiledningen vil skje via Teams

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

N/A	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

N/A	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene: N/A

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
N/A	ett år	
N/A	to år	
N/A	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:
N/A

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i sju eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

NTNU

Signaturer:

Instituttleder: Åse Ringlund Dato: 06.02.23 Åse Ringlund på vegne av inst. leder NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET
Veileder ved NTNU: Filip Holik Dato: 7.2.2023
Ekstern virksomhet: Norges Bank - avdeling Norges Bank Investment Management Birgitte Bryne (Chief Technology and Operating Officer) Guro Heimly (Legal Advisor) Guro Heiml Birgitte Bryne Dato: 31. ja 2023
Student: Celina Heimdal Brynildsen Celina H Brynildsen Dato: 6/02-23
Student: Thea Urne Thea Urne Dato: 6/02-23
Student: Sebastian Hestsveen Sebastian Hestsveen Dato: 6/02-23
Student: Anniken Arildset Anniken Arildset Dato: 6/02-23

Appendix H

Thesis description

Task description from NBIM.

Bachelor Thesis: Securing the Software Development Life Cycle

Company: Norges Bank Investment Management (NBIM)

Address: Bankplassen 2
P.O. Box 1179 Sentrum
NO-0107 Oslo, Norway

Contact person: Astri Marie Ravnaas, +47 91 69 07 85, astri.marie.ravnaas@nbim.no

Background

Securing the Software Development Lifecycle (SDLC) is about ensuring security at the different stages of software development. This includes from planning through implementation and running in production. In order to accommodate frequent deployments to production, it is important to automate the security testing by building it into the deployment pipeline. The security testing can further benefit from shift-left, where testing is done as early as possible in the pipeline. The user experience is another important aspect. How to best secure the SDLC is a large and actively developed area with a lot of interest from the industry. Systemizing the state of the art and demonstrating how certain tools and techniques fit together will have value both to NBIM and other organizations.

Goal

Create a report outlining how to best secure parts of the SDLC. We want to focus on the deployment pipeline, from submitting new code to GitHub to deploying it to AWS. It should be based on reviewing different tools, as well as implementing a proof of concept demonstrating how the different tools can be used together. The proof of concept should demonstrate how we can maintain integrity of the code throughout the pipeline, as well as scanning for security misconfiguration and vulnerabilities at key stages of the pipeline. The user experience and ability to scale to an enterprise environment should be taken into consideration.

Summary

- Use GitHub to host source code and AWS as deployment environment.
- Evaluate relevant security tools and create a proof of concept demonstrating how the tools can be integrated together.

NTNU

Norwegian University of
Science and Technology