University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Master of Science in Computer Science	Spring semester, 2021 Open / Restricted access
Writer: Bao Khanh Pham Sohrab Chalishhafshejani	..... (Writer's signature)
Faculty supervisor: Martin Gilje Jaatun External supervisor(s): Jarle Nygård, Volue Arkadiusz Doroszuk, Volue	
Thesis title: Automated software security activities in a continuous delivery pipeline	
Credits (ECTS): 30	
Key words: DevSecOps Software security CI/CD pipeline	Pages: 78 Stavanger, 15 June 2021

Front page for master thesis
Faculty of Science and Technology
Decision made by the Dean October 30th 2009

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Automated software security activities in a continuous delivery pipeline

Master's Thesis in Computer Science
by

Sohrab Chalishhafshejani

Bao Khanh Pham

Internal Supervisors

Martin Gilje Jaatun

External Supervisors

Jarle Nygård, Volue

Arkadiusz Doroszuk, Volue

June 15, 2021

"Please notice, security is not shown as a discrete part, it's just built in throughout. Ideally, security would be mostly automatic and somewhat invisible to the development and operations staff. Quite different than the way most organizations integrate security today, which is often very manual and invasive."

Aaron Levey

Abstract

Due to the rise of cyberattacks in IT companies, software security has become a topic for debate. Currently, to secure their products, companies often use manual methods, which makes development stalled and inefficient. To speed up a software development lifecycle, security work needs to be integrated and automated into the development process. This thesis will provide an initial solution for automating the security phase into a continuous software delivery process. This solution involves integrating security tools into a Github repository by using Github Actions to create automated vulnerability scanning workflows for a software project. The solution will then be tested and evaluated with three open-source projects and one project from our sponsor, Volue.

Acknowledgements

First, we would like to thank our supervisor, Professor Martin Gilje Jaatun, for his academic guidance as well as the continuous feedback that enabled us to complete this thesis.

In addition, we also thank Jarle Nygård and Arkadiusz Doroszuk, two experts from Volue, for giving us knowledge about software development as well as security in the industry.

Contents

Abbreviations

Acronym	What (it) Stands For
CI	**C**ontinuous **I**ntegration
CD	**C**ontinuous **D**elivery
GDPR	**G**eneral **D**ata **P**rotection **R**egulation
IaaS	**I**nfrastructure **a**s **a** **S**ervice
SDLC	**S**oftware **D**evelopment **L**ife-**C**ycle
QA	**Q**uality **A**ssurance
OWASP	**O**pen **W**eb **A**pplication **S**ecurity **P**roject
ASVS	**A**pplication **S**ecurity **V**erification **S**tandard
SCA	**S**tatic **C**ode **A**nalysis
CEDS	**C**yber-security for **E**nergy **D**elivery **S**ystems
RCE	**R**emote **C**ode **E**xecution
XSS	**Cross** **S**ite **S**cripting
OSS	**O**pen-**S**ource **S**oftware
TPC	**T**hird-**P**arty **C**omponents
SSH	**S**ecure **SH**ell
API	**A**pplication **P**rogramming **I**nterface
IDE	**I**ntegrated **D**evelopment **E**nvironment
DAST	**D**ynamic **A**pplication **S**ecurity **T**esting
SSL	**S**ecure **S**ocket **L**ayer
TLS	**T**ransport **L**ayer **S**ecurity

Chapter 1

Introduction

The rise of competition in software companies to deliver software with more features faster started a new way of thinking to software development. Companies try to invest in features and capabilities which are embraced and accepted by users and there is a great risk involved with developing software without immediate feedback from end-users. This has led to shortening applications' time-to-market intervals in order to receive feedback and evaluate the market faster before further development. As a result, Agile software development has been introduced to bring the development team closer to business teams and shorten the time which applications can be delivered to customers over time. This model tries to add new features on each iteration of the software life cycle and build up features optimized based on end-user needs [1].

The Agile development idea was not possible by traditional software delivery methods. In recent years software development has been shifting from delivering a finished product (software as a product) system to a software as a service principle. In this method, the software has been mostly shifted from on-premises servers to cloud solutions. Several companies have been shifting users from desktop apps to web apps. This enables providers to deploy new features on their software in short intervals without being concerned about backward compatibility issues or users being always on an updated version of the software [2].

The path to Agile development requires better cooperation between the development and operations teams in companies. Traditionally, the development team was responsible for creating logic and core features of the program and the operation teams handled deploying the app on the server/cloud. The two parties were normally only connected via ticketing systems. As a result, software delivery was delayed since most of the tasks in the operations team were manually done and problems along the way required sending and receiving several tickets between the development and operations team. This was

not enough for an Agile development process that tries to deliver software iteratively as fast as possible. DevOps was the solution. DevOps engineers are responsible to automate operations handled by the operations team such as building and testing. Automating operations processes would result in shorter operation team time to deploy changes and reach the values in Agile development and reach continuous development and delivery. Companies like Google and Microsoft first stepped into the game and introduced pre-made DevOps utilities and software which enabled faster software delivery [3].

In contrast, there are several trade-offs to this approach of software delivery. DevOps engineers try to push code batches in short intervals and it will force security teams to review the newly generated code faster. Traditionally all security analysis procedures like vulnerability scannings and code analysis were done manually by the security team. This will result in either delayed releases or a lack of security considerations by companies. Reducing security checks could result in misconfigurations, hardcoded passwords, or other dangerous concerns that can be later exploited by bad actors to breach the whole system. Furthermore, Agile teams look at security as a factor slowing their process, and oftenly, it is ignored. Cloud environments and containerization tools also add up to security holes, putting the whole process at serious risk. There is a need for faster security checks and automated testing systems which can reduce the amount of work needed by security teams manually. This will help the Agile development team to be able to check their committed software against security tools and generate instant reports on their code security issues. In addition, it helps the security team to focus more on severe vulnerabilities and act retroactively by preventing security concerns before reaching the release phase instead of waiting for vulnerability to be exposed by third parties or attackers.

Therefore, the DevSecOps concept was born to combine DevOps and Security most effectively. DevSecOps integrates an automated continuous security model with a regular Continuous Integration and Continuous Delivery (CI/CD) pipeline through vulnerability scanning tools. These tools are added to the many phases of the software continuous delivery pipeline. This automates a lot of works that were supposed to be done manually by the security team [1].

In this thesis, we are going to look for possible ways to automate software security checks in the DevSecOps procedure by building up a workflow from security tools. This workflow will then be tested with three open-source projects and one real-life project from Volue - our sponsor of this thesis. This will help companies to scale up security in the whole DevOps process with a one-time effort by changing the pipeline and integrating the security part into it.

1.1 Motivation

Based on the National Vulnerability Database dataset[1], it is clear that each year there are more vulnerabilities found; the yearly amount has doubled from 2016 until now with a peak of 18356 cases in 2020. These statistics are definitely showing an increasing trend that needs urgent action from cyber security experts. Software security measures have to be reconsidered and redefined. In October 2016, Uber faced one of the biggest data compromises of history with over 50 million Uber riders' data being breached [4]. Uber lost millions of dollars in addition to bad publicity based on data breaches which could be easily prevented by tightened security measures. The recent attacks on SolarWinds demonstrated how even monitoring and infrastructure management software can be used by bad actors to break into companies [5].

Traditionally, security has been given insufficient attention since the need for strong software security measures is not high enough on management's radar to convince them to invest more in it. However, in the case of cyber attacks or data breaches, it is visible how important this is for the reputation and survival of the whole organization. As a result, it is necessary to re-evaluate security measures in software companies. Manual security procedures are prone to human error which can be catastrophic, and manual procedures are also slowing the whole speed-to-market requirements of modern application development environments. Automating software security measures is the answer to the problem. It will speed up the whole DevOps process and avoid human intervention as much as possible.

In this thesis, we are focusing on the best solution to integrate security tests and considerations in the DevOps process to achieve higher software reliability. It is also advised by new General Data Protection Regulation (GDPR) rules that security must be taken more seriously in the fields which work with user data and keeping up with these rules is mandatory for companies servicing in the European market.

1.2 Problem Definition

The real question that arises here is what do we have to automate? There are several phases in software development that each can be a target by malicious actors. It is recommended that organizations look back at their whole infrastructure and try to understand where they can apply security measures in their DevOps environment. In addition, the infrastructure of applications is now more volatile since it is shifted more

[1] https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&search_type=all

to Cloud Native apps and infrastructure as a service (IaaS) platform. These security measures must contain ways to ensure security. Containerization and Microservice development also create new concerns which must be considered. In this thesis, we are working on how it is possible to secure web applications by applying different tools already available or creating tools needed to integrate security in the software development pipeline.

1.3 Usecases/Examples

Although it has been mentioned in recent years, the amount of academic research on DevSecOps is still meager. However, there are quite a few practitioners who have researched, experimented, and published their gray literature reports (white papers, blogs, articles, etc.). Hence, we can see that DevSecOps is getting a lot of attention from the industry [2, 6].

Besides, there is a growing choice of tools for companies to build their own pipeline. All major service providers strive to build and deliver the best platforms and frameworks for their customers. Security companies try to create tools that can be integrated with those platforms. Major platforms include Github Actions, Amazon Web Services, Microsoft Azure, Gitlab, etc. Also, some companies choose to combine different tools and platforms or develop tools for internal use.

1.4 Challenges

Although promising to bring practical results, the application of automated security testing in the CI/CD pipeline in practice also faces certain difficulties. Below are the difficulties in applying DevSecOps mentioned by Myrbakken and Colomo-Palacios [2].

- The new DevSecOps process must match the DevOps process. That means automated security checks have to be integrated into the existing CI/CD pipeline and have to be truly efficient at its speed.
- Companies face sizable volatility. That is changes about:
  - Techniques Security engineers must understand the DevOps process and developers must learn basic security skills and standards.
  - Process speed Security scanning tools are time-consuming for each new build.

  - Culture There will be a change in the culture of working at the company as security teams and product development teams combine. In addition, the understanding of security must also be disseminated to all other departments to get a truly secure process.
  - Standards New security standards will be applied and updated continuously from reputable organizations.

- Choosing the right tool for each platform is also a job that requires attention. Good tools are not necessarily suitable for the platform used by the company if they are not effectively integrated. Besides, not all tools are available to be integrated. Some tools have to be created or redeveloped from existing open-source code.

1.5 Contributions

This thesis is researched with the aim of finding an effective workflow to integrate security into the DevOps model for software companies in general or for Volue in particular. This project was proposed by Volue to be able to find a workflow that fits their workflow.

1.6 Outline

In the next chapter of the thesis, we will introduce and summarize the basic concepts in theory. Then, chapters 3 and 4 provide information about the tools that are available along with our selection. By chapter 5, our implementations, experiments and results will be presented and evaluated. Chapter 6 contains discussions of the final result of the workflow. Finally, in chapter 7, we will talk about future directions and summarize the thesis.

Chapter 2

Background

In this chapter, we are providing answers about how DevSecOps and software security automation is described and approached in literature.

2.1 Software Development Life Cycle

Software development life cycle (SDLC) is the process of a software project being developed and operated. It usually involves major stages such as planning, implementation, testing, operation, and maintenance. Some of the oldest and most popular SDLC models include the waterfall model or the V-Model [7]. However, along with the development of technology, those old models were outdated and no longer suited to the needs of the software development industry. At that time, Agile was introduced as the preeminent software development model, and later, other models were gradually developed by researchers and engineers.

2.2 Agile Software Development

The Agile development story started when a group of software consultants signed the Manifesto for Agile Software Development in 2001 [8]. Traditional methods of software development like the waterfall method did not take into account the unpredictability aspect of the development environment [9]. These methods were more focused on sequential work iterations and preparing requirements for each part of the design before moving to the next part. Testing teams were only involved in the final phase of development and problems were hidden until the testing phase. This method was not flexible to market and business. Any changes had to be completely researched and put in the next release

of development [7]. On the other hand, the Agile method was introduced and it has been the leader for many years. Agile puts user feedback in front and welcomes changes from the market or users at any time during the development procedure.

Agile meant that software development cycles were reduced and finished products could be shipped faster. This will help companies greatly since products are exposed to users faster and feedback could be gathered to polish the current development process based on user needs. There are other methodologies introduced in recent years like Kanban[10] and Scrum[1] which can be categorized as inheritors of the Agile method. Kanban focuses more on maximizing efficiency and reducing work currently in progress by each team based on their capabilities. The Scrum method tries to create short development runs called Sprints to create software and evaluate it at each iteration [11]. All mentioned methods have been used by industry for years. They are the foundation of software delivery life cycles and will play a big role in software development companies.

2.3 DevOps Methodology

The first Agile real-world implementations[8] were mainly concerned about how it is possible to improve the overall development experience. However, developers were only focusing on code delivery at the time. The path for the codebase to reach clients were taken care of by the operation team. These two teams were incomplete with different pathways of delivery ideology. Development teams were based on swift actions and changes while the operations team only focused on stability and predictability of software changes. DevOps is a unique software delivery methodology that focuses on principles and practices to bridge the gap between development and operation teams. It was introduced to enable continuous feedback and response pipelines which will also result in reduced software development cycle time [12].

The main difference between Agile and DevOps is that DevOps heavily focuses on collaboration between Development and Operations teams. In addition, it has been creating standards for pipelines and automated delivery methods which helps developers to publish their code into production with less hassle. This will lift the workforce from operation teams and make them available to monitor the product and work closely with the development team to fix problems along the way in production [3]. Automation is one of the key roles in DevOps practices and should be applied whenever and wherever possible. There are several areas in which DevOps have been constantly focusing on improvement which are mentioned below.

[1] https://www.scrum.org/resources/what-is-scrum

2.3.1 DevOps Focus Areas

In the previous section, we have identified DevOps principles like increased deployment frequencies and lower time to market. Achieving these goals requires fundamental changes in several areas in the software development environment. Here are the main focus domains which DevOps tries to improve continuously.

Teams Collaboration

In a software project, the collaboration between teams is a key to keep products stable since changes are implemented fast and on the go. Reducing delays and communication gaps is important and can only be applied by new tools and cultural change in the whole software company. Collaboration software enables development and operation teams to go beyond emails, physical meetings, and regular talks and bring on a new level of connectivity [13]. Tools like Slack, Jira, Trello, and Codesourcer are examples of collaboration technologies that are widely used in industry to connect teams that are sometimes even geographically far away from each other. However, they should be accompanied by a reform in the mindset of teams to include cooperation and teamwork culture in the employees. Regular feeds from different phases of application development like unit testing and code analysis enable all parties to detect and solve issues faster. As a result, an overall team spirit for sharing useful data will be created.

Automation Wherever Possible

In practice, phases of an SDLC are often continuous. The lifecycle is repeated over and over again, and steps like planning, developing, building, testing, or deploying the software are continuous cycles [14]. When applying DevOps to those stages, one of the most important things we need to pay attention to is optimizing the performance of the work. To do that, the automation of repetitive steps without the intervention of engineers is essential. Processes like building, testing, and deploying often repeat in large numbers, even with very small changes to the code. To handle these phases manually, the developers and operators will take a lot of time to complete. Therefore, DevOps encourages automation wherever possible. Automation saves more time during repetitive tasks, such as building and testing newly added code or modified old code [15]. By using automation, engineers can have more time for other stages that cannot be automated, such as planning, coding, or debugging.

Monitoring

Since automation is a critical DevOps goal, monitoring is indispensable for automation to follow its exact trajectory. Monitoring the system to make sure the system, pipeline, and tools are working as they should be. Once a problem occurs at any stage, it's easier and more efficient to resolve it with a carefully monitored system [16]. In fact, software logs are often cumbersome and confusing, which makes it difficult for engineers to analyze and process them. However, quite a few engineers today still use purely manual debugging tools, which results in a significant reduction in productivity. With automated assistive tools, monitoring and measurement can be better accomplished [15]. System monitoring and automation are interrelated, where monitoring makes automation more accurate, while automation makes monitoring faster.

2.3.2 CI/CD Pipeline

The CI/CD pipeline, which has the basic components Continuous integration (CI) and Continuous Delivery (CD), is basically an Agile-based pipeline for SDLC optimization. The CI/CD pipeline is intricately constructed to ensure phases of software development can be continuous. In recent times, the CI/CD pipeline has gradually become an important component in software development, making SDLC more flexible, more efficient, and faster. Finally, with the rise of cloud solutions and big cloud providers releasing command-line tools for deploying applications, continuous deployment has been added to the SDLC as a final step on the pipeline. Here we focus on each of these phases in detail.

Continuous Integration

CI is the process that allows software developers to integrate new code into the original repository as well as share them throughout the workflow. Along with that, CI automation also allows detecting any error at an early stage to commit the problem to be solved immediately when it occurs [14]. When the new code is merged with the existing repository, a new version will be activated. After the build is completed, test runs are automatically performed against the build to ensure nothing goes wrong. The integration is continuous (making it to be the "C" in CI). The build automatically verifies the code every time the developer pushes their changes to the repository. Therefore, development teams may determine problems early and have time to come up with solutions.

Continuous Delivery

Inspired by distributors and deliveries, CD is a software engineering approach based on software production in short cycles, which makes it easy for publishers to test, build, and deploy regularly. At the same time, it also reduces costs and risks when changes occur. CD is considered as an extension of the CI, and is the regular code upgrade to ensure the quality assurance (QA) [17]. The CD phase occurs at the end of the CI cycle and is responsible for the automatic distribution of the integrated code from the development stage to the production stage [14]. CD is not only tasked with automatically sending the integrated code, but also ensuring the code is sent with no errors or delays. This phase helps developers to incorporate new code into the main branch with a high degree of consistency. The CD part of the cycle is also responsible for checking code quality and performing checks to ensure a functional build can be released into the production environment.

Continuous Deployment

CI/CD process made massive changes in codebase manageable and possible during the daytime. Continuous deployment is another "CD" with a purpose beyond continuous delivery [18]. Continuous deployment tries to deliver these changes to end-users at a more accelerated speed. This approach tries to automate the deployment process and deliver up to hundreds of deploys in a day. Currently, tech giants like Facebook and Flickr have adopted this method. Software as service solutions and API-Driven[19] software facilitates projects to have daily updates hidden from the end-users [20]. In conclusion, continuous delivery releases software to deployment as soon as the tests have passed in development. It will make features time to market even lower than before.

2.4 Security In Software Development

Security has long been an indispensable part of any information technology system. When a software product is born, the development team and the customer should always consider its security. However, sometimes security is postponed in software development for different reasons. In addition, software developers have to put more time and effort to enforce all security measures in the development process. When it comes to software security, there are many possible approaches, one of them is the security of the product itself which may contain the issues that the code and builds carry full security features. To ensure this, engineers can implement a variety of methods such as periodically scanning for vulnerabilities in software, both static and dynamic code; constantly checking and

updating libraries or dependencies; make sure not to use sensitive hard-coded variables; etc. Essentially, product security is to ensure that products that are deployed will not create vulnerabilities that could compromise the system on which the product is installed. On the other hand, security problems of the development process should also be considered. They are the problems that engineers need to face to protect their own brainpower. Since systems used to develop products may be subject to attacks from cyber criminals or adversaries, engineers must ensure thorough construction of a pipeline or SDLC. This system must be considered carefully in terms of security such as confidentiality, availability, and integrity. In general, the two views above are similar to the two sides of the coin, they must coexist and support each other so that the product can be considered secure [21, 22].

2.4.1 DevSecOps

In a traditional software development process, securing a product is often done independently by the security team, separate from the development team. However, a prerequisite for DevOps is speed, the combination of the development team and the operation team and leaving the security team to work separately does not meet the needs of the industry. Hence, in recent times, the DevSecOps concept was posed as an upgrade for DevOps, when it was aimed at integrating additional security into DevOps properties [2].

DevSecOps arise with the requirement of secure output from the DevOps process. However, it is challenging in real-world implementations to introduce security to the DevOps process. Firstly, there are several toolsets available for DevSecOps. This brings inconsistency between each organization's implementation. Operations teams may be proficient in different programming languages and tools written based on them. There is currently a lack of standard DevSecOps implementation standards. This leads to different opinions between operation teams with other parts of the organization. In addition, new DevOps security tools have to be tested themselves to ensure the overall security of the organization [23].

Chapter 3

Security in CI/CD pipelines

In this chapter, we are going to discuss currently available solutions applicable to create software security pipelines. The competition is intense in pipeline tools, and many tech giants have been introducing tools trying to keep themselves in the game. While there are many similarities in how these systems work in the core, several key features differentiate these tools and give us more dependable results.

3.1 Security Requirements Of The Organization

Security baselines are a group of pre-defined configurations and checks which ensure that the development environment complies with companies' overall security policies. These policies are established by well-known tech giants or even governmental organizations to keep businesses and companies safe from cyber threats. For instance, "Microsoft Windows security baselines"[1] and the Canadian Government's "Baseline Cyber Security Controls for Small and Medium Organizations"[2] are examples of two security baselines provided by Microsoft corporation and Canadian government cyber taskforce team, respectively. Companies can have different definitions of security baseline in their environment. For instance, a company providing an online Application Programming Interface (API) for travel ticket booking may have a lower security baseline compared to a financial company handling sensitive transaction data. Security baselines can be broken into sections and applied as release gates. Release gates are critical checkpoints in an SDLC. These gates halt specific software release processes in case of reaching a security weakness threshold [24]. Introducing several release gates at once will exhaust developers by lots of

[1] https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-security-baselines

[2] https://cyber.gc.ca/en/guidance/baseline-cyber-security-controls-small-and-medium-organizations

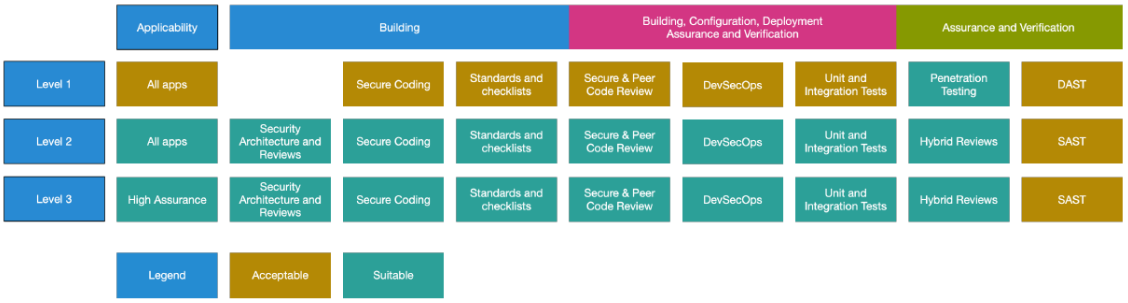

Figure 3.1: "OWASP Application Security Verification Standard 4.0 Levels" by OWASP foundation. (Creative Commons Attribution ShareAlike 3.0 license)

failed builds and releases. They have to be introduced gradually while the development team gets to know the new workflow of the pipeline. It is possible to integrate release gates in several SDLC stages like design, coding, testing, and deploying. As a result, these security checks can improve overall software quality and security in long-term usage.

3.1.1 Where To Start

Defining a security testing baseline starts by establishing minimum expectations from tests to be valid. Several industry-standard references have been already working on application security baselines in the past. The Open Web Application Security Project (OWASP) is the leading foundation supporting open source software security projects. The OWASP Application Security Verification Standard (ASVS)[3] project tries to document several in-depth verification steps to ensure application security at different security baselines. ASVS consists of 3 levels which makes it suitable for different companies with different security requirements. In this project, we are trying to cover basic ASVS recommendations which can be covered by automated tests. This includes code scanning, static and dynamic security testing, and monitoring which will be covered later.

3.2 CI/CD Pipeline Definition

Continuous Integration is the introduction of an automated system helping a team of developers in an organization. This solution tries to build, test, and validate code frequently. Regular checkups help to increase code quality, find bugs easier, and create overall better software. Continuous Delivery is the process of delivering new changes in the software to the end-users in a sustainable manner. New software features, configurations, and production versions can be released with an automated system that handles the

[3] https://owasp.org/www-project-application-security-verification-standard/

process. This will minimize the amount of time needed for our new features to hit the market. Adding the steps above will create a general pipeline. This pipeline helps the development and operations team to focus more on the new features instead of taking the heavy work of manual reviews, software deployment, and environmental setups [25]. To summarize, CI starts with a change detected in the codebase by developers. The application will be built based on the new commit and tested against unit tests. Unit tests can block unwanted or incompatible changes from processing in the pipeline. After these tests are passed, the pipeline will create an experiment environment to deploy the app. In addition, continuous delivery tests will be run in the next stage. Manual reviews can be inserted along the pipeline in case of any review needed by repository maintainers. In the end, the deployment phase begins and software will be delivered on the production server to customers [26].

3.3 Inserting Security In The Pipeline

It is possible to insert security measures alongside the pipeline mentioned above. Automated security measurements improve and assess the security of software before reaching users. The following sections present different parts of the pipeline which can be strengthened by security measures.

3.3.1 Integrated Development Environment (IDE) Plugins And Linters

Text editor applications are being used every day by developers to produce and commit code. In recent years, text editors have been empowered by plugins and third-party applications. These applications can be installed by text editors' own marketplace. This allows developers to check for quality errors while writing the code receiving live feedback from the editor. Tools created as a plugin for text editors are commonly called linters. The visual studio code editor maintains several linters like ESLint[4] and SonarLint[5]. When using linters, several bugs and security issues are prevented even before reaching the pipeline. In addition, it is possible to integrate linters in the pipeline process. Each alternative has advantages and disadvantages. Plugin linters will reduce the amount of pipeline running because it has already fixed parts of issues before reaching to pipeline and it is less possible that the pipeline fails. In contrast, not all developers can keep updated and synchronized with the linter that should be used in the entire team. This brings a risk of undetected security concerns due to obsolete and out-of-date linter versions.

[4] https://eslint.org/docs/about/
[5] https://www.sonarlint.org/features/

3.3.2 Static Code Analysis

Static Code Analysis (SCA) tries to discover vulnerabilities while the code is not in running mode. This type of analysis is often associated with white-box analysis due to access to the source code of the application. SCA uses methods like Taint Analysis[6] and Data-Flow Analysis[27]. Taint analysis tries to detect patterns related to injection vulnerabilities. It tries to identify tainted variables and traces them to possible vulnerable functions known as a 'sink' [28]. These variables need to be sanitized before handing them over. In case of improper sanitization, SCA issues a warning and informs for the possible vulnerability. These tools follow data flow and register possible output values of functions and blocks. Later on, the information collected will be used to check for predefined rules that indicate dangerous code patterns. However, methods using static code do not cover runtime vulnerabilities and it is important to test the code in the situation where it is meant to be working on. There is also a possibility of false negatives in which vulnerabilities are residing in a codebase but the tools are unable to detect it due to their limited rule-based system.

3.3.3 Dynamic Application Security Testing

Dynamic Application Security Testing (DAST) tools try to have a black box view over the application. There is no prior knowledge about the codebase or database design for these tools while running. The goal is to simulate and assess conditions in real life when the software is exposed to the network. This type of tool is developed to be running on already deployed apps in a simulated production environment. Organizations normally deploy software in specific environments via containerization applications like Docker[7] and Kubernetes[8]. Containerization tools will give a suitable environment for the app to run over particular firewall and storage configurations. DAST tools try to access the application deployed on containers over ordinary network connection means like HTTP requests or database connection requests [29]. Requests to the application can be tweaked to do a stress test on the whole application. Scenarios like exploiting, Denial of service attacks, and other known attacks can be simulated. Zed Attack Proxy[30] and Burpsuite[9] are examples of dynamic application security tools. DAST tools consist of mechanisms to bypass some security measures to go deeper into the scanning process. For instance, it is possible to bypass authentication by injecting authentication data in requests. As a result, organizations can estimate their software resistance to known attacks in case somebody acquires credentials and bypasses the authentication checkpoints.

[6] https://thecyberwire.com/glossary/taint-analysis
[7] https://www.docker.com/
[8] https://kubernetes.io/
[9] https://portswigger.net/burp/enterprise

Chapter 4

Solution Approach

This chapter will discuss the solution we have chosen for Volue. In the first part, we will talk about the current situation of Volue pipelines as well as the company's requirements for pipeline security. After that, we will then present our choices for the solution.

4.1 About Volue

Volue is one of the leading companies currently providing software and insight systems that help businesses shift to sustainable solutions. The area of focus is mainly in the energy sector with software packages facilitating energy production, trading, and distribution. In the meantime, Volue is trying to support more than 2000 customers with cloud-based IT solutions in this area. Software security is one of the main concerns in the energy sector and this has led to several efforts initializing the Cyber Security Strategy of the EU for the Energy Sector. Software security is one of the main concerns in the energy sector. The US government has established the Cybersecurity for Energy Delivery Systems (CEDS) Division to assess threats in this sector and create a preventional and response team to confront them. Furthermore, Europe has also created teams to evaluate the Cyber Security Strategy of the EU for the Energy Sector [31]. An example of cyber threats is ransomware which forced the United States to shut down the largest fuel pipeline in the country [32]. The risk is present and advanced security measures have to be in place to avoid any disruptions in this sector. Volue itself has been recently facing a cyber security accident. On May 2021 a ransomware shut down applications providing support to water plants in 200 Norwegian districts. The company acted swiftly and most systems were up and running after a short period of time [33]. This is not the first time cyber attacks happening in Norway. In 2019, Hydro, another Norwegian industry giant has also been the victim of cyber attacks [34]. These accidents show the importance of

having a strong readiness and taking software security seriously from the development phase to production and monitoring. Volue is actively evaluating its security. In this project, we try to strengthen their software development and monitoring pipeline by automating security checks whenever possible.

4.1.1 The Existing Pipelines

There are several software delivery routes currently happening in Volue extending from desktop apps to API-First[35] approaches. API-First design methodology tries to solve issues happening in the process of shifting an organization's data and operations to the cloud. This method tries to attract the attention of the development team first on the design and implementation of the application API. Later on, other interfaces are built on top and attached to the API itself. This approach reduces the amount of entanglement between the services and even software modules and makes it easier to move the apps to the cloud or containerize or scale them easier. Unification of the source control systems and build environments is an essential goal to reach software security. This will ensure security is forced and enabled in all different software development sectors. Currently, the company uses TeamCity, Azure DevOps, and Github as their main DevOps infrastructure.

4.1.2 Company Requirements

We have been appointed to improve the software security pipeline in 3 main sections, development, pre-deployment, and post-deployment phase. Volue requires a solution that has minimum configurations per repository and can easily reproduce outcomes on different pipelines. This is important in the later adoption of the work since it can be implemented with less extra weight on the development team.

4.1.3 CI Tools

The first step is choosing the right CI tool. There are several factors to consider when picking the right candidate. Jenkins, TeamCity, Travis, CircleCi, Gitlab, and Github Actions are the key competitors in the current CI tools. Each one of these products is heavily maintained and supported by its respective owners. The first important feature to be considered is the ability to host the product on-premises or on the cloud. Since all code and builds are checked and confirmed by the CI tools, the source code and keys could be in danger of being exposed. This scenario is still possible even if the product is hosted by well-known cloud hosting providers. The mechanism we chose had to be able

to provide self-hosting features for the target environment to keep all the data flow inside the company's internal network. Besides, it would be great to have the feature of cloud hosting for builds as an alternative to keep the setup and configuration complexity lower at first. At the time of writing this document, we have encountered that only Travis, Jenkins, Gitlab, and Github Actions support both on-premises and cloud hosting for pipeline builds.

The second factor considering the pipeline tool is integration and support. Later on in this chapter, we will see that some pieces like code scanning rely on third-party companies to provide the feature. Pipeline tools can be integrated with third-party software via plugins developed by the community or the company itself. These official plugins reduce the risk of including third-party apps since there is no need for developing own middlewares to connect them to the system [36].

Finally, containerization support is a must for the tool chosen for security testing. Containers enable developers to develop, test and deploy software more reliably by keeping the environment continuously the same. The CI tool must support containerization tools to test software security features and simulate the production deployment area characteristics. We have chosen Github Actions[1] as our tool since it meets all requirements mentioned above. In addition, Github Actions is based on Github repositories which currently host most of the world's open-source projects. It is possible to share each CI workflow and make it useful for the whole open source community. Github Company provides a huge amount of free build-time for open source projects. As a result, software security and integrity could drastically improve by creating automated software security pipelines over Github Actions.

4.1.4 Security Tools

Github Actions combines features of a pipeline with Github source control and repository management capabilities. Individual software developers and companies active in cyber security can release tools based on the Github Actions platform. It brings new capabilities for security players in the market to integrate their tools with the platform. In addition, all processes are made in YAML file format. YAML[2] is a structured file format that can be easily read and translated into meaningful steps. Using this file type creates an extra layer of mobility to the CI/CD process. DevOps representatives can easily define pipelines for each repository by adding a YAML workflow file to the repository. Github Actions identify the pipeline specification file and try to build the pipeline based on the exposed YAML file. Each file is considered a separate pipeline and will be initiated

[1] https://docs.github.com/en/actions
[2] https://yaml.org/

on each repository change based upon the rules defined in the file. Next, a container instance gets generated in the virtual machines hosted by either Github or On-Premises. It is possible to generate a virtual environment on Linux, macOS, and Windows-based runners.

There are several issues concerning this approach to running the environment. Firstly, when a runner starts the process, there is no way to input extra data during the process until the pipeline is done and the report is generated. It is only possible to deliver all our extra fields of data as an argument when the pipeline is starting. While running, we have encountered that some fields of data like deployment host keys, tools' API keys, and other secrets need to be injected into the application. Putting these data in the pipeline file brings a security concern of all people accessing the repository being able to have access to all our keys. An ideal scenario would be to only have the DevOps team and repository maintainers have this access. This problem was solved along the way by using an environment secrets tool. Github provides an environment secret passing tool that manages secrets and other sensitive data of the repository in a secure enclave and only passes them on demand to the runner when requested. This method will ensure that the company's sensitive information is only accessed when runners are initiated through a secure enclave. There is always a possibility of a developer trying to catch the identity of the secret providers or try to access unauthorized secrets. Currently, Github Actions is using a libsodium[3] sealed box implementation to store secrets. This library which has adopted its core functionality from the NaCl[4] library provides secure access to secrets while keeping the secret provider identity out of reach. The premise of sealed boxes is that the virtual environment has an anonymous dropbox. Github users can use a sealed box to send a message to the environment. On the recipient side, the environment knows nobody but the sender could have read or tampered with it. But it knows nothing about who the sender was. This way, we can prove the validity of the secrets without exposing where it comes from. In order to prevent unauthorized secret access, there are reviewer steps for secrets.

Each secret can have a reviewer apply access controls every time it is used during the pipeline. This may seem like an excessive controlling scenario. However, in big-scale software corporations, it is important to protect secrets like deploy API keys or database credentials out of reach of members of the organization while giving them access to use them in a pipeline to run automatic tests. It is important for organizations to use these reviewer steps to avoid the risk of an internal data breach [37]. By using all the above-mentioned techniques we will ensure that the virtual environment and secrets will be kept secure while running in the organization.

[3] https://libsodium.gitbook.io/doc/public-key_cryptography/sealed_boxes
[4] http://nacl.cr.yp.to/

4.1.5 Code Scanning & Dependency Checking

Codebases are always prone to include vulnerabilities that are easily detectable by machines. Although there could be possible logical vulnerabilities in applications, it is possible to prevent others by automated pipelines. There have been several comparisons between SCA tools before which we can see examples from Matti Mantere; Ilkka Uusitalo; Juha Röning [38]. In addition, Arvinder Kaur and Ruchikaa Nayyar also compared these tools in [39]. These comparisons try to reach these tools mostly based on vulnerabilities found and how they try to fix them automatically. In previous years, several tools are released in this area and some of the comparisons mentioned have been changed with new updates to the tools. In addition, while these measures are important, another key factor for our project was how much it is possible to integrate the solution to an automated workflow of the Github Actions ecosystem. Furthermore, new characteristics like duplicate code detection are only available in recently introduced versions of tools like SonarQube. In this project, we have focused on tools with a focus on integration and extension of their scanning capabilities. We have tried several tools like SonarQube, Snyk, and Fortify; the results will be presented in the following chapters. There is not a clear winner since each tool is presenting better features in different sectors. SonarQube provides extensive capabilities in detecting code smells and providing code review while Snyk is better at dependency scanning and security measures. We are looking to reach two goals in this part of the pipeline. First providing a code quality scan for the repository we are operating on. We have measured that are quantitative and qualitative to examine our code submission and move forward along the pipeline.

Code Coverage

Code coverage tests try to calculate lines of code used while running test suites. It is normally measured by a percentage which shows what percentage of the current codebase is used by tests. This test is essential to ensure software security. Each line of the code being tested by a Unit test provides a level of confidence that the following code functionality has been examined in tests. This will prevent software developers from creating intentional or unintentional loopholes in the system which has not been tested during software development. Examples of this behavior could be developers creating extra functions which log data created in the enterprise and save them in a remote machine [40]. If this part of the system has not been visited by the tests, the Code coverage number will decrease. It is possible to set a threshold in the pipeline to fail decreasing coverage code commits. Finally, we can consider hardcoding minimum code coverage percentage which is usually set around 80 percent when looking at discussions in DevOps forums. It is not suitable for projects which start to adopt these new security

pipelines since it will break the whole pipeline by not letting any code commit pass the tests if the previous codebase is weakly tested. As a result, we recommend setting rules to improve code coverage percentage on each commit based on previous reports' results. SonarQube provides code coverage results which we can further process and threshold our pipeline based on the results provided.

Code Smells

Code smells are not essentially bugs or vulnerabilities but they ring an alert about the codebase overall code quality. An example of code smells is empty defined functions in the program. Code smells to try to make an assessment of the future technical debt in the system based on the templates defined for them. Having a code smell in the developers' commit is not an immediate problem and it can be considered just as a warning during the pipeline. We will try to consider SonarQube as our code smell provider tool. The pipeline will only generate a report on the cod smells and not try to prevent developers from finishing the pipeline if any code smell exists. Finally, the generated report is presented to the repository maintainer to decide on suitable actions [41].

Code Quality Checker

Unlike code smells which do not convey a serious problem in our codebase, Code Quality checkers look for dangerous patterns in our code that may expose applications to possible vulnerabilities like remote code execution (RCE), SQL injection, and cross-site scripting (XSS). Projects may vary in codebase size from thousands of lines to millions of lines. The bigger size of the codebase brings up new complexity to the code and makes it harder to review. Code quality checkers are tools that define sets of rules which get checked against the whole codebase to find possible breaches in the rules. This will ensure policies are enforced in the whole codebase. In addition, human manual reviews are prevented at this stage since humans are more prone to mistakes when it comes to inspection of big complex codebases. Quality warnings will be generated later based on the level of severity of the problem and it is up to the maintainer of the codebase to let the whole pipeline process on a different level of code quality breaches. In some modern code quality services, each issue comes with a recommended solution to fix it manually or by running commands to solve automatically [42].

Dependency Scanning

Software supply chain security is currently under study and development due to the recent events in the cyber security world. After the Orion network monitoring hack which affected big firms like US Homeland Security and Microsoft, there is a major focus on regulating dependencies used in projects. Dependencies are data or functionality imported into the software by external sources. Developers and maintainers have little to no authority over the dependencies and monitoring their behavior need a meticulous inspection of the dependency functionality and design. Open-source software (OSS) development is becoming mainstream. We are relying more on third-party-developed software every day. According to a survey by Black Duck Software, 78 percent of responders reported that their whole company or parts of it is using OSS [43]. In addition, 66 percent of participants indicated that their company produces software based on OSS. Third-party components (TPC) and dependencies are used as building blocks of the software. this enables companies with less human and financial resources to build software faster while relying on the extensive library of TPCs over the internet [44]. Although many of the TPCs are being inspected and tested by volunteers, they are still considered as black boxes which companies inheriting them have no idea of what happens inside the TPC. Here we face a paradox. A tool that using it can advance the development process and help us to build better and with a blazing speed. On the other hand, the same TPC can act as an attack surface to our final product and bring on new opportunities for malicious actors to penetrate our software. Eventually, TPCs are being imported to our project using a centralized registry. Malicious actors can use these imported libraries and software to infiltrate and exploit the internal systems of the organization. It is possible to prevent this threat by using dependency scanners. These scanners try to keep track of vulnerabilities or security holes in the TPC's and inform the software maintainer about it. Furthermore, deprecated and obsolete dependencies are spotted and removed from the software lifecycle. It is possible to keep dependencies automatically updated to get security patches as soon as TPC's are updated. We have decided to use Snyk as our dependency scanner. Snyk has integrations with Github Actions and provides extensive dependency checking with an updated database of vulnerable packages. Package managers like Pip, Npm, Go and Composer are crawled on a daily basis and security vulnerabilities are added to Snyk Vulnerability Database. These vulnerabilities range from simple buffer overflow possibilities to higher degrees of dangerous vulnerabilities like arbitrary code execution or privilege escalation [45]. Snyk tries to score the found vulnerabilities based on the severity of the problem. As a result, we have been able to produce a roadmap of the software codebase and problems alongside how critical and time-dependent is to fix that issue before causing bigger problems in the software lifecycle.

4.1.6 Open-port Scanning

It is possible to send requests to a range of ports in the server to find active ports. Port scanning is one of the oldest tactics used by hackers to gather information about which services and applications are running on the target. Enterprise applications normally use known ports to connect to external applications. Although some ports are vital for the application to be running, a few ports are only used for maintenance objectives. For instance, port 22 is used for Secure Shell (SSH) access to provide access for server maintenance. These ports can be left open while they are not in use and pose a threat. Attackers can try to exploit less secure applications using their special port. Besides, organizations are expanding the connectivity of devices with IoT solutions in the business. This will introduce even more network-connected machines which can have misconfigured applications and additional open ports [46]. Volue has given us the assignment to have an automated port scanning solution to provide 24/7 surveillance over the deployed applications in the Volue ecosystem. This will enable Volue to get notified whenever a new port has been opened on the applications without prior whitelisting. We have used Github Actions to provide monitoring capabilities. A script will run NMap[5] application at different time intervals during the day. NMap is a free open source software that is widely used by network administrators to perform network discovery and auditing. We have used NMap software in our automated pipeline to watch over Volue deployed apps and report which ports are currently open. Later on, Volue can use this data to detect misconfigurations or internal mistakes which lead to open exposed ports in their network.

4.1.7 Secure Socket Layer And Transport Layer Security (SSL/TLS) Evaluation

Creating a connection between server and client over the internet needs security measures to avoid eavesdropping and packet modification. Secure Socket Layer and Transport Layer Security protocol try to achieve this goal by providing an authenticated and encrypted channel between two parties with novel algorithms and cryptographic ciphers. This method tries to use public/private key encryption. To keep the public keys accessible to others certificate authorities provide essential data alongside public keys needed to connect to that server. However, in recent years we have faced several security breaches in SSL/TLS protocol. An example of the security leak in SSL is the Heartbleed bug[6] which allowed malicious actors to steal critical documents, emails, messages, and other valuable information from users who still use older versions of OpenSSL software. Besides, currently, only a few TLS versions and cipher suites are approved by governments and

[5] https://nmap.org/book/man.html
[6] https://heartbleed.com/

industry experts to be used [47, 48]. Volue has mentioned that it is important for them to keep SSL/TLS updated and secure. We have been working on the pipeline to provide security checks to check for SSL/TLS version and configuration. This test will be part of the pipeline and can be run automatically in time intervals. Volue can include all server endpoints of the company in the mentioned pipeline and we will do evaluations of the server configurations, SSL/TLS versions, expiry date of certificates, and overall security measures of the encrypted connection. SSLLabs[7] scanner is command-line opensource client software that creates tests for hosts' SSL security. We will be integrating SSLLabs software in our command line and generate reports from command line results of the software so that it can be reviewed by repository owners. It is also possible to stop the pipeline when a misconfiguration has been detected in SSL/TLS security. This will work as an always-on watching system to protect volue SSL/TLS encryption from possible attacks due to misconfigurations and outdated versions of software.

4.1.8 Integration And Automation

Github Actions use virtual machine runner instances to run pipelines. Each pipeline occupies portions of the runner's computational power and bandwidth usage. It is important for our automated pipeline to be efficient and only run tests and builds whenever needed. It is possible to run tests on special conditions and Github runners will initiate the action whenever the trigger conditions are met. Software companies consist of different teams working on the product at the same time. If all the pipeline tests run on every push to the repositories, this will exhaust the runners, and jobs will be in long queues before they could be completed. In addition, specific tests like SSL/TLS scans are time-consuming and will keep the runners busy for longer periods. We tried to develop smart pipelines that try to expand the runner's idle time by assigning pipeline triggers to persons committing the code. As an example, user experience and UI design teams' commits are ignored by runners. Developers' commits will trigger static analysis, code coverage, and code smell pipelines. SSL/TLS scans and port scans work parallelly in time intervals without attachment to commits. Finally, commits to master will run all possible runners to check for everything before the new code is added to the system [49]. These are just examples of roles in a software organization and how they can be assigned to pipeline triggers. We have the ability to tweak the system so that different jobs happen in special circumstances. As a result, pipeline triggers will be controlled and runner machines will not be under pressure.

[7] https://github.com/ssllabs/ssllabs-scan/

Chapter 5

Pipeline Evaluation

We will explain how we install and operate our pipeline in different environments in this chapter. First, we used 3 open-source projects that we found on the internet to test the different tools we chose. Finally, we integrate several suitable tools into Volue's repository for a complete security workflow that can integrate into the CI/CD pipeline.

5.1 Experimental Systems Setup

In this thesis, we prepare three projects that test each part of the pipeline, along with a project repository provided by Volue in the middle of the research period to test the performance of the complete pipeline. The three repositories we use are open source projects that can be found on the internet in the three languages required by Volue, namely Python, Javascript and C#. The practical project powered by Volue is written in C# with the DotNet framework.

5.1.1 Test Repositories

We use three projects with three different languages, Python, Javascript and C# respectively. The first two projects we used to test tools with scripting languages. We then use a project written in the DotNet framework in C# - a compiled language, from which to build a suitable pipeline for Volue's project.

Python Repository

This is a simple project written in Python and Flask which is the first project we used to test Github Actions as well as apply CodeQL to check vulnerabilities in the project.

Since CodeQL is provided by Github with a feature that only works in public open-source projects, we make this repository public at the link https://github.com/sohrabch/github-actions

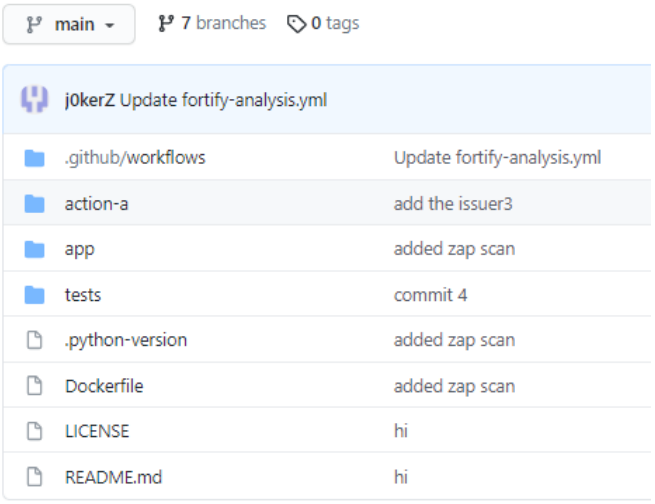

Figure 5.1: Python repository

We mainly use this repository to get acquainted and use Github Actions to create a workflow. Besides, we also test the vulnerability scanning tools here.

Javascript Repository

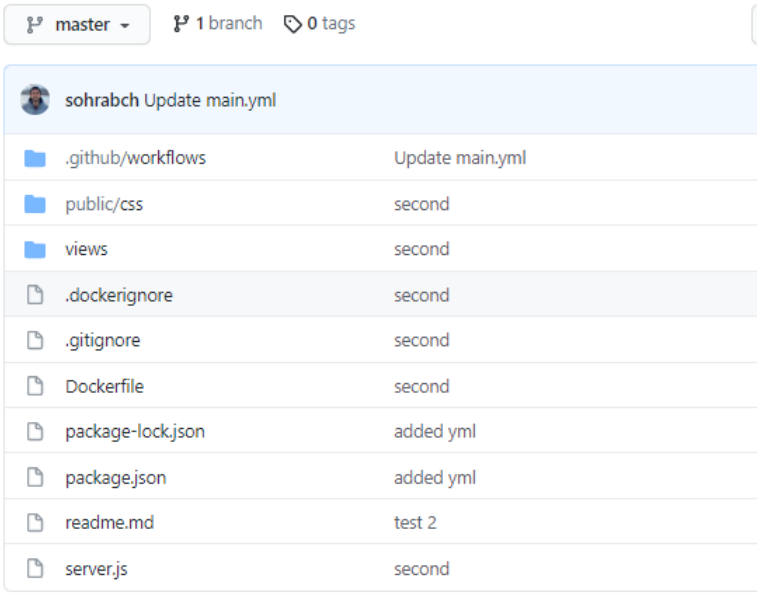

Figure 5.2: Javascript repository

This is a simple weather application written in Javascript that we cloned from https://github.com/bmorelli25/simple-nodejs-weather-app#readme. We use this repository to develop our pipeline to work more coherently, trying to break it down clearly so that we can understand how the tools work comprehensively.

Here, we build a series of steps using Github Actions to test Snyk, njsscan, as well as other open port and SSL scanning tools. From there, combine them to become a sequence of security scans in a CI/CD Pipeline.

C# Repository

This repository is taken by us from Bitwarden - an open-source project written in C# with DotNet framework which used to store login and password information. This is an actual project being operated and contributed by the community. This project includes many smaller components such as server, web, desktop and phone applications, and so on. We use the Bitwarden server codes to test our pipeline. The source code of the repository can be found at the following link: https://github.com/bitwarden/server

This repository we install as private to be able to test as close as possible to Volue's project. Different from the previous two languages, this project uses C#, which is a compiled language. Therefore, the vulnerability scanners for the project also requires the code to be compiled before calling the tools. This makes each scan time on this project longer than in scripting languages such as the two above.

5.1.2 Volue Repository

This is an actual project of Volue, this project is written with DotNet and C#. For security reasons as well as Volue policy, the source code of this project cannot be published. At this project, we run the final stages of testing for the pipeline we built. In addition, we also perform upgrades and modifications to tools or pipelines by the demand from Volue.

5.2 Implementation

Our pipeline, like all Github Actions workflows, is placed in a yml file located in the *.github/workflow* directory. In this file, we use the tools by one or more steps in the workflow.

167 lines (139 sloc) | 5.23 KB

```yaml
name: Security test
on:
  push:
    branches: ["master"]

jobs:
  Full-security-scan:
    name: Build and security scan
    runs-on: ubuntu-latest
    strategy:
      matrix:
        dotnet-version: ['3.1.x']
    steps:
    - name: Checkout repository
      uses: actions/checkout@v2
```

Figure 5.3: yml file header

The figure 5.3 shows a header of a yml file. The **name** part is used to name the workflow, this field is used to distinguish when the pipeline has several different threads. Next, the **on** item is used to set the condition under which the workflow is triggered. Usually, there will be two main types here, event-based or time-based. In our experiments, we mostly set up the workflow to start up on **push** events. The goal is that with each push coming from the development team, the code will be automatically scanned for vulnerabilities once to detect dangerous errors early.

Workflow steps are located in the **jobs** section. Each job will have its own name at the beginning and the name must not contain spaces. The next section is **run-ons**, where the operating system on which this job will run is indicated. Finally, the steps in a job will be declared in the **steps** section.

For each project, there will be different steps defined and organized based on user intent. However, to be able to interact with the project's code, the Checkout step must always be called first. At this step, the Github Actions system will clone the entire repository into the virtual system on which the job is running. This step is written as the figure 5.4.

```yaml
steps:
  - name: Checkout repository
    uses: actions/checkout@v2
```

Figure 5.4: Checkout step

Python Repository

We have used this repository to test a few different tools like CodeQL, Fortify or ZAP. However, we will only talk about the tool that gives the most stable results that make us

most satisfied, which is CodeQL. This is a tool provided by Github security engineers themselves. So we do not need to use too many implementation steps to use it.

26 lines (18 sloc) | 508 Bytes

```yaml
name: Security check for python-test project

on: [push, pull_request]

defaults:
  run:
    working-directory: "test_projects/python-test"

jobs:

  CodeQL-scanning:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: CodeQL Init
        uses: github/codeql-action/init@v1
        with:
          languages: python
          token: ${{ secrets.ACTIONSTOKEN }}
      - name: CodeQL analyze
        uses: github/codeql-action/analyze@v1

```

Figure 5.5: Action steps with Python project

First, the **init** step must be defined along with the language used by the project. This step helps the tool know the language being used to install the necessary libraries and plugins. Then we simply invoke the **analyze** action and wait for the results to be returned in the **Security** tab at the very repository where the workflow is running.

Javascript Repository

In this repository, we tested a few tools namely Snyk, njsscan, NMap and SSLLabs. In addition, we also use the results obtained and uploaded with the actions themselves as artifacts.

```
59 lines (56 sloc)  1.76 KB
1   name: Example workflow for Node using Snyk
2   on: push
3   jobs:
4     security-check-snyk:
5       runs-on: ubuntu-latest
6       steps:
7         - uses: actions/checkout@master
8         - name: Run Snyk to check for vulnerabilities
9           uses: snyk/actions/node@master
10          continue-on-error: true
11          env:
12            SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
13          with:
14            args: --severity-threshold=high
```

Figure 5.6: Snyk steps with NodeJS

With a project written in NodeJS, to scan for vulnerabilities with Snyk, we need to call the actions developed by the Snyk team at *snyk/actions/node*. However, the thing to add here is that to be able to call Snyk, we have to create a Snyk account at snyk.io and get our own token. Then, put that token in the secrets section of the settings of the repository to use it in the actions. The results of the scan will be shown in the results of actions. Snyk tries to cross-reference the code with its local database of vulnerabilities and dangerous patterns in order to create a risk map and vulnerability list.

```
18    node-js-code-scan:
19      needs: security-check-snyk
20      runs-on: ubuntu-latest
21      name: njsscan code scanning
22      steps:
23        - name: Checkout the code
24          uses: actions/checkout@v2
25        - name: nodejsscan scan
26          id: njsscan
27          uses: ajinabraham/njsscan-action@master
28          with:
29            args: ". --sarif --output results.sarif || true"
30        - name: Upload njsscan report
31          uses: github/codeql-action/upload-sarif@v1
32          with:
33            sarif_file: results.sarif
```

Figure 5.7: njsscan steps

Using njsscan is quite similar to Snyk, we also call the action from the tool's repository. However, this tool does not require a tool user token. In addition, we install a tool that outputs the results as sarif files - a file format for reporting results with Github's Security tab. Then we use the upload sarif action defined by the Github team in the CodeQL repository.

The open port scanner we use here is NMap. This tool has no built-in support for Github Actions, so we installed it using a bash script in the very environment created for the workflow. Here, we run NMap to scan the Volue homepage, however, this is due to the

limited access we had to company infrastructure and deployed softwares behind firewalls. In real world scenarios, volue DevOps team will use tokens to inject deployed software endpoint to NMap software. NMap will later try to check on the endpoint ports and report on the open ports found.

```
35    port-checker:
36      needs: node-js-code-scan
37      runs-on: ubuntu-latest
38      name: port Scanner + SSL QL Check
39      steps:
40        - name: Checkout the code
41          uses: actions/checkout@v2
42
43        - name: setup Golang
44          uses: actions/setup-go@v2
45          with:
46            go-version: "^1.13.1"
47        - name: run port Scanner
48          run: |
49            sudo apt-get install nmap
50            nmap www.volue.com > logs.txt
51        - name: SSL Quality Checker
52          run: |
53            git clone https://github.com/ssllabs/ssllabs-scan.git
54            go run ./ssllabs-scan/ssllabs-scan-v3.go www.volue.com >> logs.txt
55        - name: upload artifact
56          uses: actions/upload-artifact@v2
57          with:
58            name: SSL QL JSON
59            path: ./logs.txt
```

Figure 5.8: NMap and SSLLabs steps

Finally, we use SSLLabs to evaluate SSL quality. This tool also does not support Github Actions, so we continue to install it manually through a combination of bash scripts and actions. First, we install the tool's environment which is Go programming language through actions provided by Github team. Then we cloned the tool's code to the working environment. From there, we ran the tool with Go as if we were running it on a real machine. SSL Labs checks for SSL certificate tree and certificate variables in different operating systems and devices. This is done by SSLLabs virtual environments that simulate Android, IOS and Desktop devices with different browsers.

The results of both NMap and SSLLabs tools are stored in a file named logs.txt and uploaded as an artifact with the output of the workflow.

C# Repository

In this repository, we mainly focus on perfecting our pipeline, testing Snyk when running against DotNet projects. First, to be able to run Snyk with projects that use DotNet, the project's code must be built before the Snyk action is invoked. To build it, the first step, the DotNet framework has to be installed, we do this with the actions provided by

the Github team. Then the dependencies must also be installed using DotNet with the appropriate config file. Next, we invoke the build command with DotNet to output the complete builds.

```yaml
name: Security test
on:
  push:
    branches: ["master"]

jobs:
  Code-scanning-with-snyk:
    name: Static scan (code)
    runs-on: ubuntu-latest
    steps:
    - name: Checkout repository
      uses: actions/checkout@v2
    - name: Setup .NET Core SDK ${{ matrix.dotnet-version }}
      uses: actions/setup-dotnet@v1.7.2
      with:
        dotnet-version: ${{ matrix.dotnet-version }}
    - name: Install dependencies
      run: dotnet restore --configfile NuGet.Config
    - name: Build
      run: dotnet build --configuration Release --no-restore

    - name: Run Snyk to check for vulnerabilities
      uses: snyk/actions/dotnet@master
      continue-on-error: true
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
      # Change solution file name here
      # Comment 2 lines below if not use solution file
      with:
        args: --file=bitwarden-server.sln --json-file-output=vuln.json
```

Figure 5.9: Steps to run Snyk with DotNet project

The next steps, similar to the NodeJS project, are vulnerability scanning with Snyk, open port scanning with NMap, and SSL evaluation with SSLLabs.

Volue Repository

In this repository, we repeat what we did in the C# repository. In addition, we perfect our pipeline by generating the completed output and preventing malicious pushes according to the results received.

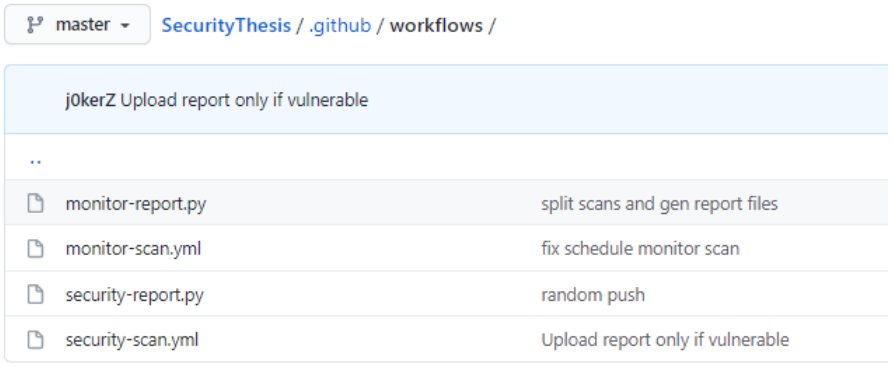

Figure 5.10: Workflow files in the Volue repository

We split the pipeline into two different components: vulnerability scanning and monitor scan which contains open port scanning and SSL certificate checking.

We scan for vulnerabilities in the code after every push coming from the developers. If serious vulnerabilities are discovered in the code, the pipeline will automatically revert the newly pushed commit.

43 lines (35 sloc) | 986 Bytes

```
1  name: Monitor security scan
2  on:
3    schedule:
4      # Once every hour
5      - cron: "* */1 * * *"
6
7  jobs:
8    Monitor-security-scan:
9      name: Monitor security scan
10     runs-on: ubuntu-latest
11
```

Figure 5.11: Schedule the workflow to run once every hour

Monitor scan will be periodically scanned depending on the request from Volue, here we set the scan to take place once every hour. The final result is summed up with a piece of python we code ourselves. After that, it will be uploaded to our Slack channel.

The code of the workflows is attached in the appendix A.

5.3 Experimental Results

We document the results from each tool as well as the performance of our pipeline to ensure the working of the automation we built.

5.3.1 Python Repository

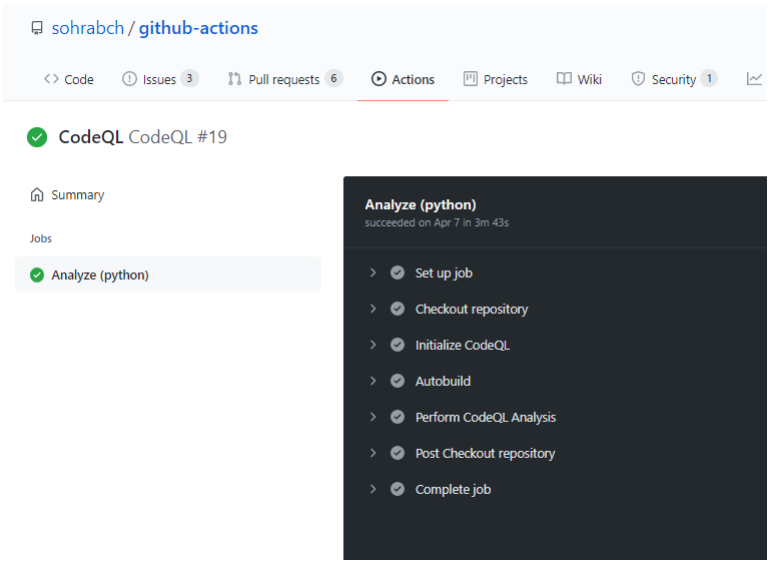

Figure 5.12: CodeQL workflow finished fluently

After much tweaking and testing, the final result we got from the CodeQL tool was quite satisfactory. The tool runs stably, does not appear error when running, and returns the correct results if a vulnerability is detected. We can see a sample result of action as shown in figure 5.12, which is the result of a vulnerability scan using CodeQL. Using artifacts helped us to integrate vulnerability reports to Github further. We can notify developers and maintainers of found issues as soon as the report is complete. It is possible to use an internal Github notification system to make this process even more integrated.

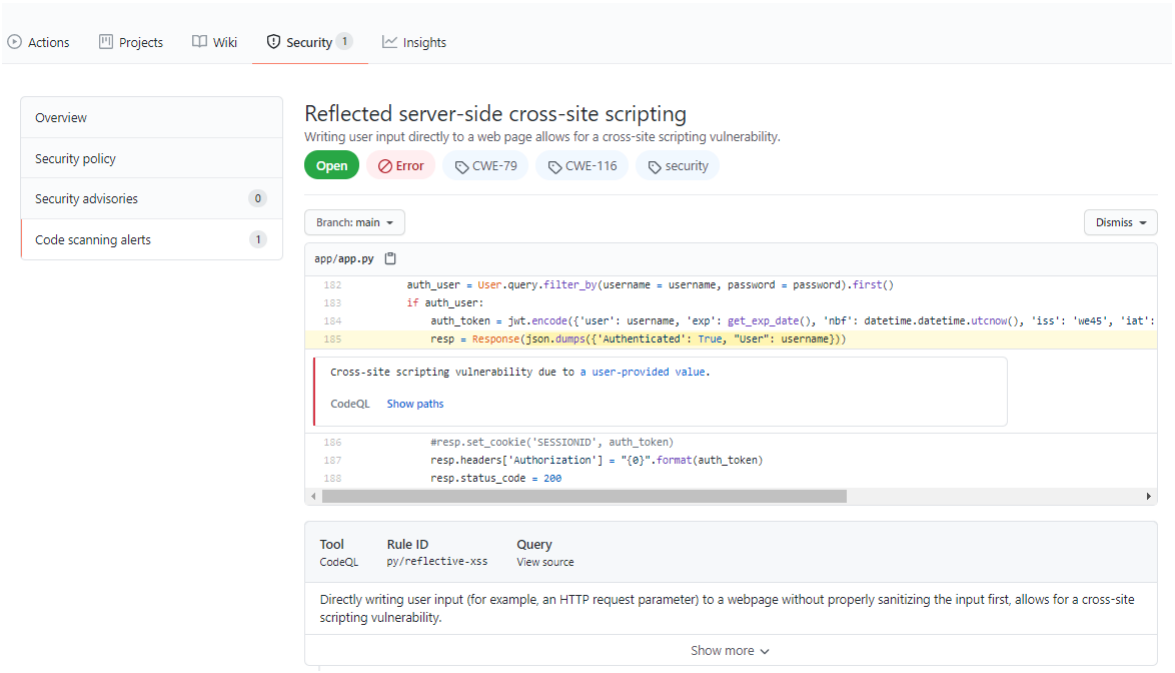

Figure 5.13: Vunerability found by CodeQL

The figure 5.13 represents the vulnerability that the CodeQL tool found in the Security/-Code analysis tag of the repository. From this extremely detailed result, the development team can patch the discovered vulnerability more easily. In addition, codeQL provides the vulnerability index in the Common Weakness Enumerations database [1] or National Vulnerability Database [2]. These are worldwide databases to submit and measure security weaknesses in software and hardware. They are also used as a baseline for weakness identification, mitigation, and prevention efforts. As a result, developers can check each vulnerability and look at community made fixes for the issue.

5.3.2 Javascript Repository

At this repository, each tool returns results in a different way. More specifically, figure 5.14 shows the results of Snyk's vulnerability scan, which is shown in the output of the scan step in the workflow.

[1] https://cwe.mitre.org/index.html
[2] https://nvd.nist.gov/

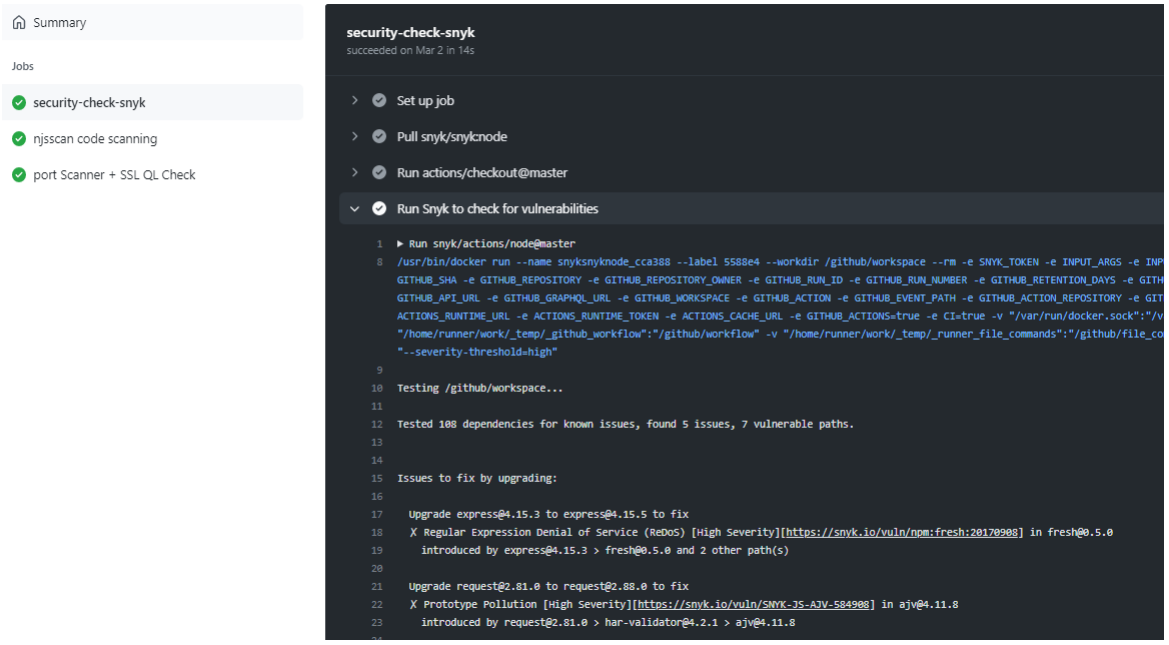

Figure 5.14: Result of Snyk

The review by Snyk vulnerability scanning takes place smoothly. The vulnerabilities found are shown in the output of the action itself. We can see the statistics of vulnerabilities discovered by Snyk in figure 5.14.

The results of njsscan can be found in the Security tab of the repository as shown in figure 5.15. We can see that the result of njsscan is different from the one obtained from Snyk.

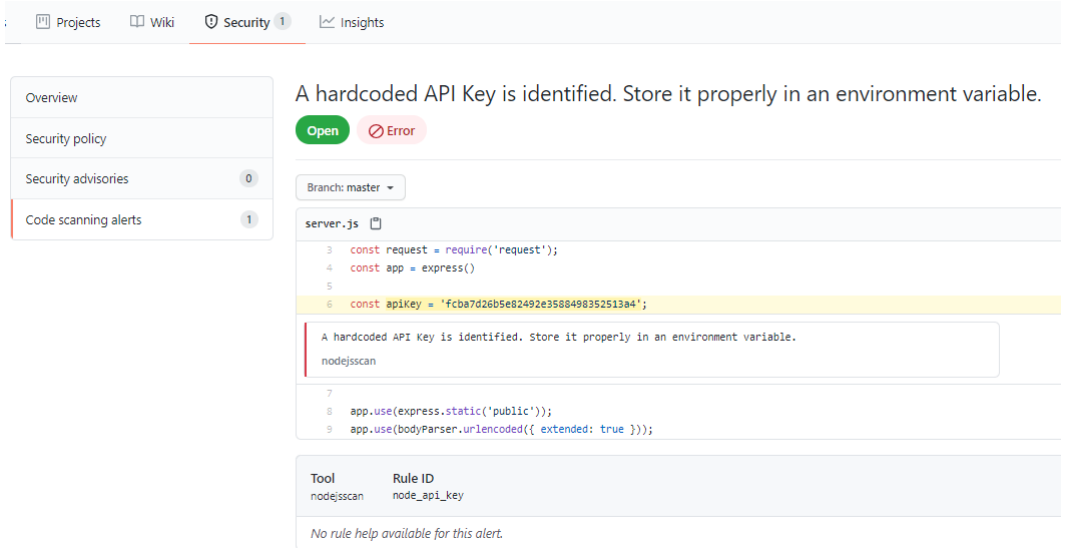

Figure 5.15: Result of njsscan

As we can see, similar to CodeQL, njsscan outputs a *sarif* file, which is supported by Github. Hence, the results are shown in the Code scanning alert list in detail. However, Github only offers this feature on public repositories or in organizations that have purchased Github Enterprise program.

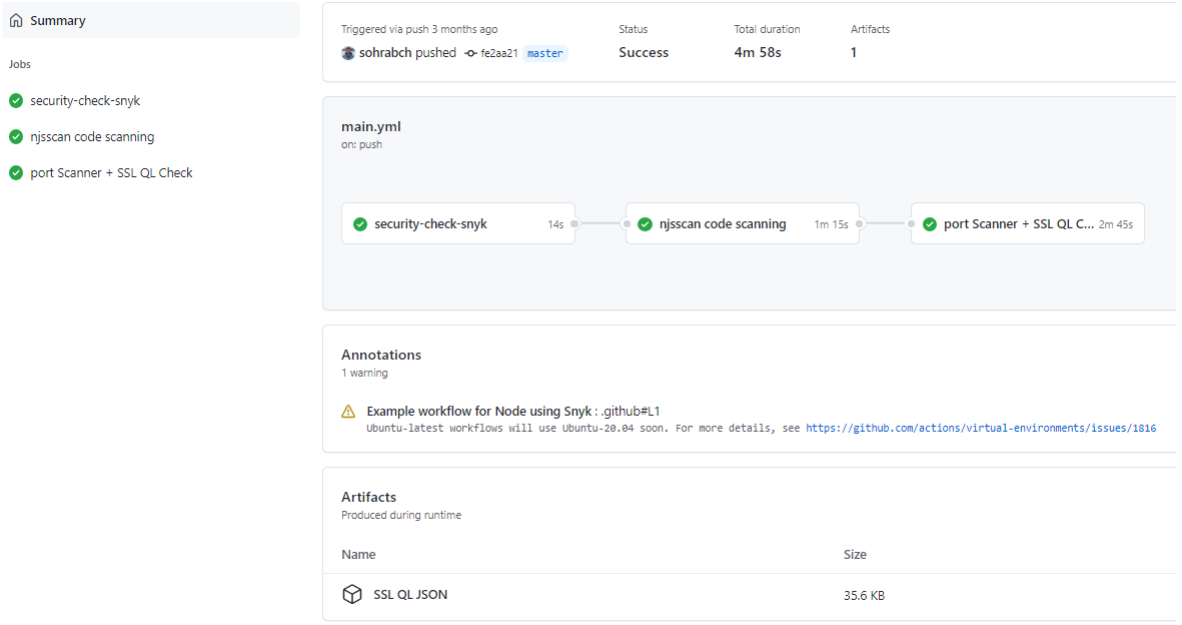

Figure 5.16: Open ports and SSL results was uploaded as artifact

The results of the open port scan and SSL check are compiled into a text file and uploaded with the action as an artifact. Figure 5.16 shows where the artifact is located. When downloaded and unzipped, we will get the result as shown in figure 5.17.

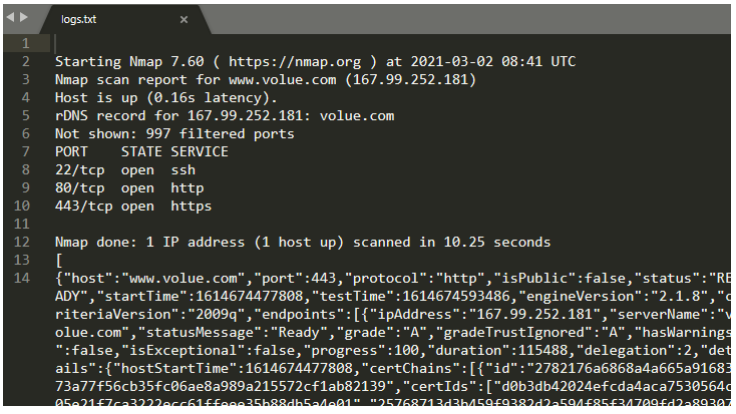

Figure 5.17: Results of open ports and SSL

5.3.3 C# Repository

Figure 5.18 below shows the results of the Snyk vulnerability scanning workflow on the Dotnet project. Before getting the scan results, we must let the system install the necessary dependencies for the project. The results are also shown in the output of the jobs.

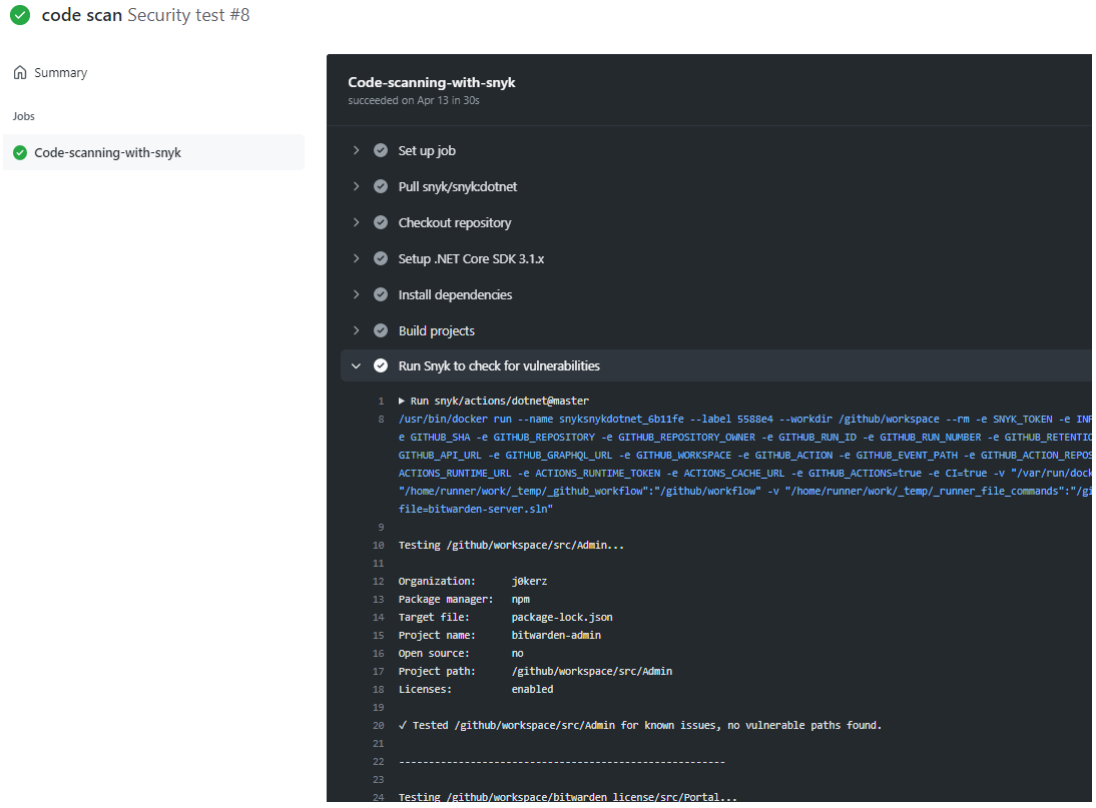

Figure 5.18: Results on C#

5.3.4 Volue Repository

Here, we get the results of two workflows at two different places, submitting to Slack and using Github Actions artifact. This can be easily customized according to the purpose and requirements of the user.

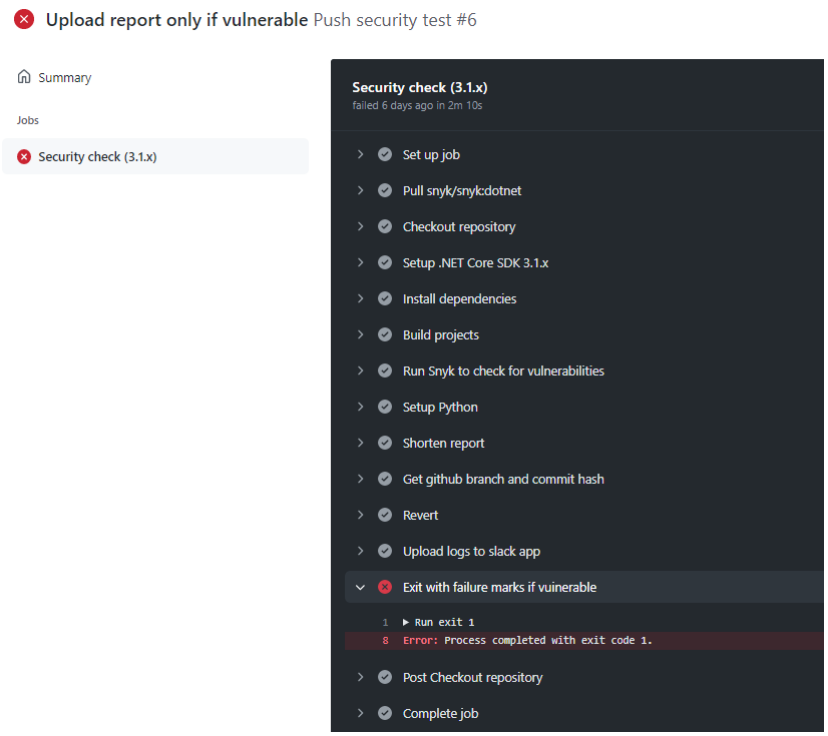

Figure 5.19: Results of security scanning on Volue repository

First, as we can see, figure 5.19 shows when Snyk discovered a vulnerability in the project. At this point, the report will be generated and uploaded to our Slack channel. The results in figure 5.20 clearly show which branches and commits have been vulnerable, as well as which vulnerabilities have been discovered. Slack is a choice of the team working on this project. We have made the report accessible over other workplace tools like Jira, Slack, and social messenger tools. This enables developers to choose their preferred way of receiving reports about their submissions.

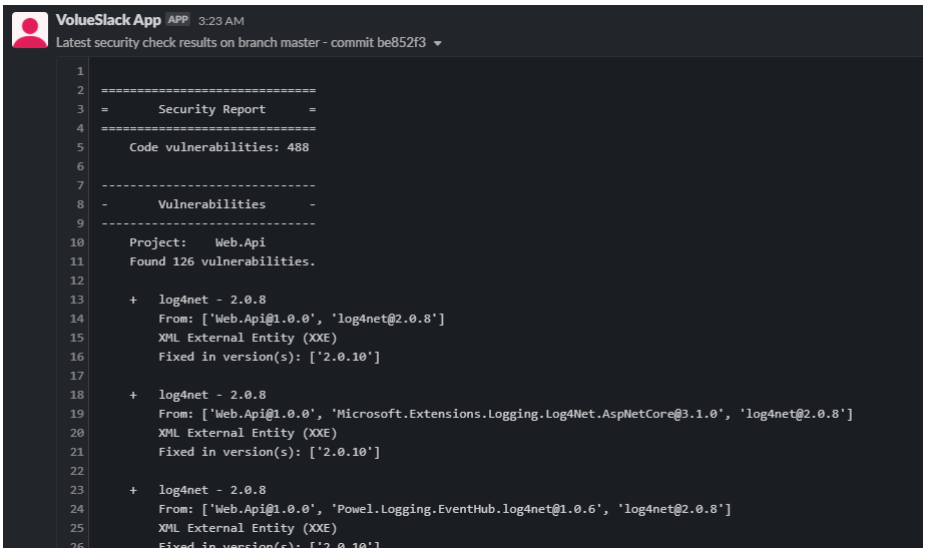

Figure 5.20: The results are uploaded to the Slack channel if any vulnerability is found

The monitor scan, on the other hand, returns the result as an artifact of the action. From the results obtained, the development team can assess whether the system is operating properly. Here, we do not include a standard to assess whether the system is safe or not because this is completely based on the policy of each company and organization. Reporting whenever there are signs of a vulnerability can be done just as easily as the vulnerability scan above.

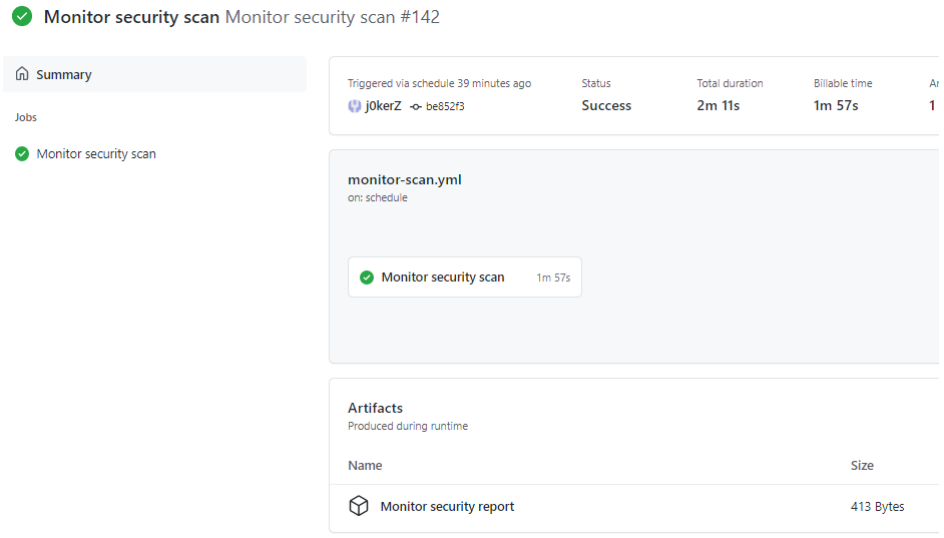

Figure 5.21: Results of monitor scan

Figure 5.21 shows us the result of the monitor scan as an artifact of the action. Along with that, figure 5.22 shows the result when decompressed. Here, we can see the SSL and open port parameters of the system that have been deployed.

```
1
2  ==============================
3  =       Security Report      =
4  ==============================
5      Open ports: 3
6      SSL Grade: A
7      SSL Certificate(s): 3
8
9  ------------------------------
10 -         Open ports         -
11 ------------------------------
12 PORT    STATE SERVICE
13 22/tcp  open  ssh
14 80/tcp  open  http
15 443/tcp open  https
16
```

Figure 5.22: The artifact file of the result

Chapter 6

Discussion

In this chapter, we are going to deliberate about the results that we have from the final pipeline for Volue. Then, we will consider the value of the pipeline with the requirements from the company.

6.1 Expectation Vs Results

Our ultimate goal in this thesis is to integrate the security component into the existing DevOps pipeline. Besides, we also wanted to automate it in the best and most suitable way for the development team. This can save time and other resources that are used for security testing. Furthermore, it also makes it easier to react to vulnerabilities, accidentally created by a developer through bad code. Instead of waiting for the penetration testing team to periodically test the project, this scan will be automatically performed and notified to the developer as soon as the code is committed. It is also expected that this pipeline can be integrated minimum amount of changes to the Volue SDLC.

In chapter 5, we were able to see the test results of the pipeline we developed. Overall, the results we have achieved are quite satisfying. The tools work stably as we expected. They are run completely automatically with only a few steps of installation that are not too complicated. When developers work, they don't have to wait for the security team to respond. Vulnerabilities will be automatically detected by the tools and reported to the development team. This helps speed up the product development process. Instead of waiting for hours or even days to detect an unintentional vulnerability, when applying this pipeline, the discovery of these errors takes only a few minutes.

6.2 The Solution For The Industry Challenges

Roshan N. Rajapakse, Mansooreh Zahedi, M. Ali Babar, and Haifeng Shen have investigated and collected challenges related to adopting DevSecOps in organizations [23]. We try to face these challenges and measure how our security pipeline product has helped to overcome them. Here we discuss a few challenges and how we managed to solve them in our pipeline.

6.2.1 Tool Selection Challenge

DevOps consists of several stages which have fundamental differences in their workflow. There are many tools developed by the industry to support each stage and include security in the pipeline workflow. The above-mentioned research shows that developers struggle in tool selection when it comes to DevSecOps implementation. This has been an issue during our project development also. Each tool comes with a feature set and implementation tactic that usually differs from the competition. In addition, tools use different programming languages to implement their features. This means that even developers inside a single organization may be in contradiction about the tool they are keener to use [50]. Our proposed solution focus on choosing the platform which has wider adoption by the community. Github is widely adopted by the developer's community and many open source projects are maintained in Github repositories. Using Github Actions as a base tool gave us the possibility to reach a wider spectrum of possible users of the pipeline. Furthermore, we tried to use official plugins from vulnerability scanners instead of writing our own bash scripts. This would help us to keep the pipeline-specific code smaller and avoid problems faced in the future to maintain the pipeline. However, there is a lack of standardization in this area to help DevSecOps teams in the implementation stages. We believe that in the coming years there would be an urge to create and maintain standards for tools developed in this area that would lead to possible guidelines for DevSecOps tools selection. There are other concerns like the security of the tools itself which are beyond the scope of this project to be evaluated.

6.2.2 Static Analysis Tools Limitations

Software dependencies are increasing and a library can be used on thousands of different dependencies. Current static analysis tools run into the problem of high false positives and a high volume of vulnerability detections in tests due to nested dependencies [51, 52]. This will exhaust developers and make it hard to find and prioritize vulnerabilities to fix. We have solved this issue by providing an option for the DevSecOps team to choose

the severity of their target vulnerability and get reports based on their selection. As a result, developers can focus on critical vulnerabilities first and lower the amount of severity whenever they have covered more critical problems.

6.2.3 Access Management In DevOps Domain

DevOps team has access to the production environment and software critical data like secrets and API keys. This is due to the definition of the team to handle deployments and overall software running infrastructure. There is always an existing threat concerning insider misuse of the keys or data in an organization. Keys can be used to access sensitive customer data or process unauthorized requests. Restricting access to production-sensitive data while keeping the workflow running is one of the challenges in DevSecOps operations. We have used Github secret management tools to overcome this challenge while giving access to secret keys in the command line tools and the runner's environment. Maintainers can issue access to secrets on-demand and DevOps members will only acquire the key usage access without accessing the original key content.

6.3 Satisfaction Of Volue's Requirements

Our workflow has gained recognition from Volue. Jarle Nygård, our local supervisor from Volue, has the following views:

"I have applied the workflow in one other project now and it was easy to set up and put to work. It has already paid off, in the sense of finding 3 high severity issues in our dependencies. The vulnerabilities were deep in the dependency graph and so it would have been almost impossible for anyone to manually find the unpatched dependencies, but with the report output from the workflow, it was easy to identify and fix the problems. My recommendation to Volue R&D leadership is to enforce the usage of this workflow for all Volue products going forward."

Jarle is a Senior Software Architect of Volue who worked with us during the time we were working on this thesis. He also gave us lots of helpful advice and some practical directions when we got stuck. As having been following since the beginning, he totally understands the workflow that we have built.

6.4 Limitations

The current system still has quite a few limitations that can be overcome in the future. First, the number of selected tools is not diverse. For now, we've only included a few tools that stand out and match the requirements from Volue. Besides, we currently only support three languages, C#, Javascript and Python. Also, current tools can only be integrated with Github Actions and not with other CI tools yet. In addition, monitoring tools must be optimized to bypass existing firewalls and regional blocks issued by companies to be able to monitor the organizations running applications.

6.5 Challenges

Since the beginning of this thesis, we have also encountered a few problems that made the research and experiment not go as expected. These problems have gradually been overcome by us in the process of completing the thesis.

The first obstacle that can be mentioned is the social distancing during the epidemic, which makes it impossible for us to go to the office and work directly with the Volue development team. Therefore, sometimes we misunderstood their ideas. However, we have overcome this by creating regular meetings as well as creating our own Slack channel for communication.

Second, when working with Github and Github Actions, we ran into obstacles due to Github's policy not to support some tools when run in a private repository. That contradicts Volue's policy that the code used to test the tools is not allowed to be released to the public. To deal with this, we have been constantly finding, testing, and replacing different tools to find the ones that can support private repositories.

Because of some of the procedures based on the policy of Volue, we were not able to experiment with actual company projects from the start. This led us to decide to find and use a few open source projects from the internet as we mentioned in chapter 5.

One last small thing that can be mentioned here is the technical hurdle, as this is the first time we approach a project written in C# or using Github Actions to create a workflow. We have learned how to use and work with those technologies, and that will help us a lot in the future.

Chapter 7

Future Directions and Conclusion

This is the final chapter of the thesis. We will sketch out some directions to further develop the solution and conclude our work on the pipeline.

7.1 Future Directions

In chapter 6, we mentioned some limitations of the current workflow. Here, we will talk about the solution and some positive directions in the future.

7.1.1 Tool Diversification

The increase in the variety of tools makes it possible for users to have more options to suit the requirements and purposes of each project. Since different tools may give different results, using multiple scanning engines simultaneously will help avoid missing vulnerabilities.

7.1.2 Increase Compatibility

The current system needs to be modified and upgraded to support more programming languages and frameworks. In addition, the development of workflow to be able to work on platforms other than Github is also a desirable thing. Depending on the nature of each different project, the deployment and running of the software will take place in different environments. Therefore, the vulnerability scanning of each case should also be carefully considered during the implementation phase.

7.1.3 DAST Further Implementation

We have been limited by the resources we have access to for dynamic security testing. Volue has several running projects which need defined testing forks and branches to apply dynamic security testing. In the future, volue can contribute to this project by developing further dynamic security features like ZAP scanner integration. We have already researched the DAST tools and tried them on the test programs. However, for DAST to work, there needs to more internal knowledge about the application workflow and authentication process.

7.1.4 Enhance Automation

Now the automation of the workflow has been partially applied. However, when deploying each project, the development team still has to install it manually. Developing a tool to automate the generation and integration of workflows based on user requests is an extremely promising direction.

7.2 Conclusion

To conclude, in this thesis, we studied how to automate security so that it can be integrated into the CI/CD pipeline. We studied the existing theories to build a workflow model from security tools for the software development process. We then used that workflow to apply to a trio of test software projects and an actual one coming from Volue. The results obtained from testing can indicate the workflow that we have shown works well for software projects and gives certain effects on the performance of a software development life cycle. The workflow we built was also appreciated by Volue - the company that proposed this thesis. Although there are still imperfections and needs to be improved, the project shows a lot of potentials to be developed into a tool to support DevSecOps work.

List of Figures

Appendix A

Workflows code

A.1 security-scan.yml

```
name: Push security test
on: [push]

jobs:
  Push-security-scan:
    name: Security check
    runs-on: ubuntu-latest
    strategy:
      matrix:
        dotnet-version: ['3.1.x']
    steps:
    - name: Checkout repository
      uses: actions/checkout@v2
      with:
        fetch-depth: 0

    - name: Setup .NET Core SDK ${{ matrix.dotnet-version }}
      uses: actions/setup-dotnet@v1.7.2
      with:
        dotnet-version: ${{ matrix.dotnet-version }}

    - name: Install dependencies
      run: dotnet restore --configfile nuget.config

    - name: Build projects
      run: dotnet build --configuration Release --no-restore

    - name: Run Snyk to check for vulnerabilities
      uses: snyk/actions/dotnet@master
      continue-on-error: true
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
      # Change solution file name here
      # Comment 2 lines below if not use solution file
```

```
      with:
        args: --file=Powel.Melding.sln --json-file-output=vuln.json

    - name: Setup Python
      uses: actions/setup-python@v2

    - name: Shorten report
      id: results
      run: |
        python ./.github/workflows/security-report.py

    - name: Get github branch and commit hash
      id: vars
      shell: bash
      run: |
        echo "##[set-output name=branch;]$(echo ${GITHUB_REF#refs/heads/})"
        echo "::set-output name=sha_short::$(git rev-parse --short HEAD)"

    # Runs these if the push is vulnerable
    - name: Revert
      if: contains(${{ steps.results.outputs.stdout }}, "Vulnerable")
      continue-on-error: true
      run: |
        cd "${GITHUB_WORKSPACE}"
        git config --global user.name "github-actions[bot]"
        git config --global user.email "github-actions[bot]@users.noreply.github.com"
        git revert --no-edit ${{ steps.vars.outputs.sha_short }}
        git push

    - name: Upload logs to slack app
      if: contains(${{ steps.results.outputs.stdout }}, "Vulnerable")
      uses: adrey/slack-file-upload-action@master
      with:
        token: ${{ secrets.SLACK_TOKEN }}
        path: ./Security_report.txt
        title: "Latest security check results on branch ${{steps.vars.outputs.branch}} - commit ${{ ste
        channel: random

    - name: Exit with failure marks if vuinerable
      if: contains(${{ steps.results.outputs.stdout }}, "Vulnerable")
      run: |
        exit 1
```

A.2 security-report.py

```
import json

vuln_file = open("vuln.json")
vuln_data = json.load(vuln_file)

report = open("Security_report.txt", "w+")
```

```
report.write('''
==============================
=       Security Report      =
==============================
''')

vuln_report = ""
vuln_count = 0
for project in vuln_data:
  vulns = len(project["vulnerabilities"])
  vuln_count += vulns
  if vulns > 0:
    vuln_report += "\tProject: \t" + project["projectName"] + "\n"
    vuln_report += "\tFound " + str(vulns) + " vulnerabilities. \n"
    for vuln in project["vulnerabilities"]:
      vuln_report += "\n"
      vuln_report += "\t+\t" + vuln["name"] + " - " + vuln["version"] + "\n"
      vuln_report += "\t \tFrom: " + str(vuln["from"]) + "\n"
      vuln_report += "\t \t" + vuln["title"] + "\n"
      vuln_report += "\t \tFixed in version(s): " + str(vuln["fixedIn"]) + "\n"
    vuln_report += "\n------------------------------\n"

report.write("\tCode vulnerabilities: " + str(vuln_count) + "\n")

report.write('''
------------------------------
-       Vulnerabilities      -
------------------------------
''')
report.write(vuln_report)

if vuln_count > 0:
  print("Vulnerable")
```

A.3 monitor-scan.yml

```
name: Monitor security scan
on:
  schedule:
    # Once every hour at xx:30
    - cron: "30 */1 * * *"

jobs:
  Monitor-security-scan:
    name: Monitor security scan
    runs-on: ubuntu-latest

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2
```

```
    - name: setup Golang
      uses: actions/setup-go@v2
      with:
        go-version: "^1.13.1"

    - name: run port Scanner
      run: |
        sudo apt-get install nmap
        nmap www.volue.com > nmap.txt

    - name: SSL Quality Checker
      run: |
        git clone https://github.com/ssllabs/ssllabs-scan.git
        go run ./ssllabs-scan/ssllabs-scan-v3.go www.volue.com >> ssl.txt

    - name: Setup Python
      uses: actions/setup-python@v2

    - name: Shorten report
      id: results
      run: |
        python ./.github/workflows/monitor-report.py

    - name: upload artifact
      uses: actions/upload-artifact@v2
      with:
        name: Monitor security report
        path: ./Monitor_report.txt
```

A.4 monitor-report.py

```
import json

nmap_file = open("nmap.txt", "r+")
nmap_line = nmap_file.readline()
while not nmap_line.startswith("PORT"):
  nmap_line = nmap_file.readline()

ssl_file = open("ssl.txt")
ssl_data = json.load(ssl_file)

report = open("Monitor_report.txt", "w+")

report.write('''
==============================
=       Security Report      =
==============================
''')

nmap_report = ""
nmap_count = 0
while not nmap_line.startswith("\n"):
```

```
  nmap_report += nmap_line
  nmap_line = nmap_file.readline()
  nmap_count += 1

report.write("\tOpen ports: " + str(nmap_count-1) + "\n")

ssl_report = ""
ssl_count = 0
for ssl_cert in ssl_data[0]["certs"]:
  ssl_count += 1

report.write("\tSSL Grade: " + str(ssl_data[0]["endpoints"][0]["grade"]) + "\n")
report.write("\tSSL Certificate(s): " + str(ssl_count) + "\n")
report.write('''
------------------------------
-         Open ports         -
------------------------------
''')
report.write(nmap_report)

report.write('''
------------------------------
-      SSL Certificates      -
------------------------------
''')
```

Bibliography

[1] Rakesh Kumar and Rinkaj Goyal. Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (adoc). *Computers & Security*, 97:101967, 2020.

[2] Håvard Myrbakken and Ricardo Colomo-Palacios. DevSecOps: a multivocal literature review. In *International Conference on Software Process Improvement and Capability Determination*, pages 17–29. Springer, 2017.

[3] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of DevOps concepts and challenges. *ACM Computing Surveys*, 52(6), November 2019. ISSN 0360-0300. doi: 10.1145/3359981. URL https://doi.org/10.1145/3359981.

[4] Eric Newcomer. Uber paid hackers to delete stolen data on 57 million people, 2017. URL https://www.bloomberg.com/news/articles/2017-11-21/uber-concealed-cyberattack-that-exposed-57-million-people-s-data.

[5] Sean Peisert, Bruce Schneier, Hamed Okhravi, Fabio Massacci, Terry Benzel, Carl Landwehr, Mohammad Mannan, Jelena Mirkovic, Atul Prakash, and James Bret Michael. Perspectives on the SolarWinds incident. *IEEE Security Privacy*, 19(2): 7–13, 2021. doi: 10.1109/MSEC.2021.3051235.

[6] Runfeng Mao, He Zhang, Qiming Dai, Huang Huang, Guoping Rong, Haifeng Shen, Lianping Chen, and Kaixiang Lu. Preliminary findings about DevSecOps from Grey Literature. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 450–457, 2020. doi: 10.1109/QRS51102.2020.00064.

[7] S Balaji and M Sundararajan Murugaiyan. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management*, 2(1):26–30, 2012.

[8] Max Rehkopf. Manifesto for agile software development, 2001. URL https://agilemanifesto.org/.

[9] Kuda Nageswara Rao, G Kavita Naidu, and Praneeth Chakka. A study of the Agile software development methods, applicability and implications in industry. *International Journal of Software Engineering and its applications*, 5(2):35–45, 2011.

[10] Chun-Che Huang and Andrew Kusiak. Overview of Kanban systems. *International Journal of Computer Integrated Manufacturing*, 9(3):169–189, 1996. doi: 10.1080/095119296131643.

[11] Max Rehkopf. Kanban vs. Scrum: which Agile are you?, 2020. URL https://www.atlassian.com/agile/kanban/kanban-vs-scrum.

[12] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. What is DevOps? a systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, pages 1–11, 2016.

[13] Vijeth Hegde and Abhijeet Singh. Team collaboration in DevOps: Accenture, Jul 2020. URL https://www.accenture.com/us-en/blogs/software-engineering-blog/hegde-singh-team-collaboration-in-devops.

[14] Manish Virmani. Understanding DevOps & bridging the gap from continuous integration to continuous delivery. *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, pages 78–82, 2015.

[15] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. DevOps. *IEEE Softw.*, 33(3):94–100, May 2016. ISSN 0740-7459. doi: 10.1109/MS.2016.68. URL https://doi.org/10.1109/MS.2016.68.

[16] Theo Schlossnagle. Monitoring in a DevOps world. *Commun. ACM*, 61(3):58–61, February 2018. ISSN 0001-0782. doi: 10.1145/3168505. URL https://doi.org/10.1145/3168505.

[17] Lianping Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32, 03 2015. doi: 10.1109/MS.2015.27.

[18] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017. doi: 10.1109/ACCESS.2017.2685629.

[19] Harish Goteti. API Driven development , bridging the gap between providers and consumers. 2015.

[20] Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, and Michael Stumm. Continuous deployment at Facebook and OANDA. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 21–30, 2016.

[21] Hala Assal and Sonia Chiasson. Security in the software development lifecycle. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 281–296, Baltimore, MD, August 2018. USENIX Association. ISBN 978-1-939133-10-6. URL https://www.usenix.org/conference/soups2018/presentation/assal.

[22] Asoke Talukder, Vineet Maurya, Babu Santhosh, Ebenezer Jangam, Sekhar Muni, Jevitha Kp, Saurabh Samanta, and Alwyn Pais. Security-aware software development life cycle (SaSDLC) - processes and tools. pages 1 – 5, 05 2009. doi: 10.1109/WOCN.2009.5010550.

[23] Roshan Rajapakse, Mansooreh Zahedi, Muhammad Ali Babar, and Haifeng Shen. Challenges and solutions when adopting DevSecOps: A systematic review. 03 2021.

[24] Alfred Chung. How DevOps can use quality gates for security checks, 2018. URL https://blog.rapid7.com/2018/05/30/how-devops-can-use-quality-gates-for-security-checks-2/.

[25] Thorsten Rangnau, Remco v. Buijtenen, Frank Fransen, and Fatih Turkmen. Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 145–154, 2020. doi: 10.1109/EDOC49727.2020.00026.

[26] S.A.I.B.S. Arachchi and Indika Perera. Continuous Integration and Continuous Delivery pipeline automation for Agile software project management. In *2018 Moratuwa Engineering Research Conference (MERCon)*, pages 156–161, 2018. doi: 10.1109/MERCon.2018.8421965.

[27] Jorrit Kronjee, Arjen Hommersom, and Harald Vranken. Discovering software vulnerabilities using data-flow analysis and machine learning. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450364485. doi: 10.1145/3230833.3230856. URL https://doi.org/10.1145/3230833.3230856.

[28] Aditya Kurniawan, Bahtiar Saleh Abbas, Agung Trisetyarso, and Sani Muhammad Isa. Static taint analysis traversal with object oriented component for web file injection vulnerability pattern detection. *Procedia Computer Science*, 135:596–605, Jan 2018. ISSN 1877-0509. URL https://www.sciencedirect.com/science/article/pii/S1877050918315230.

[29] Ferda Özdemir Sönmez and Banu Günel Kiliç. Holistic web application security visualization for multi-project and multi-phase dynamic application security test results. *IEEE Access*, 9:25858–25884, 2021. doi: 10.1109/ACCESS.2021.3057044.

[30] Simon Bennetts. Owasp zed attack proxy. *AppSec USA*, 2013.

[31] Recommendations for the European Commission on a European Strategic Framework and potential future legislative acts for the energy sector. Technical report, Energy Expert Cyber Security Platform (EECSP), 2017.

[32] Collin Eaton, James Rundle, and David Uberti. U.S. pipeline shutdown exposes cyber threat to energy sector, 2021. URL https://www.wsj.com/articles/u-s-pipeline-shutdown-exposes-cyber-threat-to-energy-sector-11620574464.

[33] Vigleik Takle. Volue after the cyberattack: How we passed the stress test, 2021. URL https://www.volue.com/news/volue-after-the-cyberattack.

[34] Bill Briggs. Hackers hit Norsk Hydro with ransomware. the company responded with transparency, 2019. URL https://news.microsoft.com/transform/hackers-hit-norsk-hydro-ransomware-company-responded-transparency/.

[35] Mario Dudjak and Goran Martinović. An API-first methodology for designing a microservice-based backend as a service platform. *Information Technology And Control*, 49(2):206–223, 2020. doi: 10.5755/j01.itc.49.2.23757.

[36] Vijeth Hegde and Abhijeet Singh. Comparison of most popular Continuous Integration tools: Jenkins, TeamCity, Bamboo, Travis CI and more, 2019. URL https://www.altexsoft.com/blog/engineering/comparison-of-most-popular-continuous-integration-tools-jenkins-teamcity-bamboo-travis-ci-and-more/.

[37] Deba Prasead Mozumder, Md.Julkar Nayeen Mahi, and Md Whaiduzzaman. Cloud computing security breaches and threats analysis. *International Journal of Scientific and Engineering Research*, 8:1287 – 1297, 07 2017.

[38] Matti Mantere, Ilkka Uusitalo, and Juha Roning. Comparison of Static Code Analysis tools. In *2009 Third International Conference on Emerging Security Information, Systems and Technologies*, pages 15–22, 2009. doi: 10.1109/SECURWARE.2009.10.

[39] Arvinder Kaur and Ruchikaa Nayyar. A comparative study of static code analysis tools for vulnerability detection in C/C++ and JAVA source code. *Procedia Computer Science*, 171:2023–2029, 2020. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2020.04.217. URL https://www.sciencedirect.com/

science/article/pii/S1877050920312023. Third International Conference on Computing and Network Communications (CoCoNet'19).

[40] Chris Burns. What is code coverage and why it should not lead development. URL https://capgemini.github.io/testing/What-Is-Code-Coverage-and-Why-It-Should-Not-Lead-Development/.

[41] Thanis Paiva, Amanda Damasceno, Eduardo Figueiredo, and Cláudio Sant'Anna. On the evaluation of code smells and detection tools. *Journal of Software Engineering Research and Development*, 5(1):7, Oct 2017. ISSN 2195-1721. doi: 10.1186/s40411-017-0041-1. URL https://doi.org/10.1186/s40411-017-0041-1.

[42] Jens Knodel and Matthias Naab. *How to Perform the Code Quality Check (CQC)?*, pages 95–104. Springer International Publishing, Cham, 2016. ISBN 978-3-319-34177-4. doi: 10.1007/978-3-319-34177-4_9. URL https://doi.org/10.1007/978-3-319-34177-4_9.

[43] Black Duck Software. Survey on open source software usage. URL https://www.blackducksoftware.com/about/news-events/releases/seventy-eight-percent-of-companies-run-on-open-source-yet-many-lack-formal-policies-to-manage-legal-operational-and-security-risk.

[44] Managing security risks inherent in the use of third-party components. Technical report, SafeCode, 2017. URL https://safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf.

[45] Snyk. Detailed information and remediation guidance for known vulnerabilities., 2021. URL https://snyk.io/vuln/.

[46] Craig Badrick. Defending against port scan attacks, 2019. URL https://www.turn-keytechnologies.com/blog/article/defending-against-port-scan-attacks/.

[47] Cloudflare. Why use TLS 1.3? | SSL and TLS vulnerabilities, 2021. URL https://www.cloudflare.com/en-gb/learning/ssl/why-use-tls-1.3/.

[48] Aaron Russell. SSL/TLS best practices for 2021, 2021. URL https://www.ssl.com/guide/ssl-best-practices/.

[49] Github. Events that trigger workflows, 2021. URL https://docs.github.com/en/actions/reference/events-that-trigger-workflows.

[50] Hasan Yasar and Kiriakos Kontostathis. Where to integrate security practices on DevOps platform. *International Journal of Secure Software Engineering (IJSSE)*, 7 (4):39–50, 2016.

[51] James Kupsch, Barton Miller, Vamshi Basupalli, and Josef Burger. From continuous integration to continuous assurance. pages 1–8, 09 2017. doi: 10.1109/STC.2017.8234450.

[52] Nora Tomas, Jingyue Li, and Huang Huang. An empirical study on culture, automation, measurement, and sharing of DevSecOps. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–8. IEEE, 2019.