

**Aatu Laitinen**

**Ensuring development efficiency with DevSecOps: A case  
study on streamlining dependency vulnerability  
management with Dependabot**

Master's Thesis in Information Technology

June 6, 2024

University of Jyväskylä

Faculty of Information Technology

**Author:** Aatu Laitinen

**Contact information:** `laitinaa@jyu.fi`

**Supervisor:** Tommi Mikkonen

**Title:** Ensuring development efficiency with DevSecOps: A case study on streamlining dependency vulnerability management with Dependabot

**Työn nimi:** Kehitystehokkuuden varmistaminen DevSecOps:lla: Tapaustutkimus riippuvuushaavoittuvuuksien hallinnan sujuvoittamisesta Dependabotin avulla

**Project:** Master's Thesis

**Study line:** Information and Software Engineering (Tieto- ja Ohjelmistotekniikka)

**Page count:** 59+6

**Abstract:** With the ever increasing need for modern software development companies to be able to continuously release new code and the increased emphasis on the security of the software, the practice of shifting security processes to the earlier stages of the development has become coveted. However creating a DevSecOps environment where the responsibility of the security processes is shifted to the developers without hindering their ability to efficiently produce software is challenging.

This work conducts a multivocal narrative literature review to research both academic and grey literature for what type of challenges the shift-left security introduces for the development speed. The review also identifies various solutions that can be utilized to mitigate the hindrance on the development efficiency. One of these solutions is the use of dependency management bots to automatically create fixes for vulnerabilities in projects' dependencies. This solutions is further studied on by implementing it to a real world company environment in the form of a case study.

As a part of the case study a guideline was created for how GitHub's Dependabot can be used to speed up the dependency vulnerability fixing process. The case study used a Likert-scale questionnaire to gather insight and prejudice on the presented usage of Dependabot.

The results indicated that there is a gain in terms of increasing the speed of the vulnerability fixing process as well as increasing the overall security of the projects. The identified barriers for the implementation were also deemed in the results to not be restricting factors for the adaption of the Dependabot's security updates. The small sample size of the study and the limited view of only a single company means that the results can not be used to reflect the global view on the matter, but the study's results can still be utilized as an entry point for Dependabot's security update adaptation.

**Keywords:** DevSecOps, shift-left, DevOps, Dependabot, development, efficiency, security

**Suomenkielinen tiivistelmä:** Nykypäivän ohjelmistokehitysyriyten kasvava tarve kyetä jatkuvasti julkaisemaan uutta koodia sekä ohjelmistojen turvallisuuden entistä suurempi korostuminen on johtanut siihen, että tietoturvaprosessien siirtämisestä kehityksen aikaisempiin vaiheisiin on tullut oleellisempaa näille yrityksille. DevSecOps-ympäristön luominen, jossa tietoturvaprosessien vastuu on siirretty kehittäjille siten, ettei heidän kykynsä tuottaa tehokkaasti uutta koodia heikentyisi, on kuitenkin haastavaa.

Tässä työssä toteutetaan moniääninen narratiivinen kirjallisuuskatsaus, jonka avulla etsitään vastauksia sekä akateemisesta että harmaasta kirjallisuudesta sille, millaisia haasteita tietoturvan siirtäminen vasemmalle aiheuttaa ohjelmistokehitysnopeudelle. Kirjallisuuskatsauksessa tunnistetaan myös erilaisia ratkaisuja, joilla voidaan lieventää tätä kehityksen tehokkuuden heikentymistä. Yksi näistä ratkaisuista on riippuvuuksien hallintaan kehitetyt botit, joiden avulla voidaan automaattisesti päivittää projektien haavoittuvuudet riippuvuudet. Tätä ratkaisua tutkitaan myös tarkemmin toteuttamalla se tosielämän yritys ympäristössä tapaustutkimuksen muodossa.

Osana tapaustutkimusta luotiin ohje siitä, miten GitHubin Dependabottia voidaan käyttää nopeuttamaan projektien riippuvuuksien haavoittuvuuksien korjausprosessia. Tässä tapaustutkimuksessa käytettiin Likert-asteikko-pohjaista kyselyä näkemyksien ja ennakkoluulojen keräämiseen esitetyn Dependabotin käytön suhteen. Tulokset osoittivat, että haavoittuvuuksien korjausprosessin nopeutta ja projektien yleistä turvallisuutta on mahdollista parantaa toteutuksen avulla. Myös tunnistetut esteet Dependabotin tietoturvapäivitysten käyttöönotolle eivät tulosten perusteella olleet tätä rajoittavia tekijöitä. Tutkimuksen pieni otoskoko

ja rajoittuminen vain yhteen yritykseen tarkoittaa, että tuloksia ei voida pitää yleispätevinä, mutta tutkimuksen tuloksia voidaan silti käyttää lähtökohtana Dependabotin tietoturvapäivitysten käyttöönotolle.

**Avainsanat:** DevSecOps, shift-left, DevOps, Dependabot, kehitys, tehokkuus, tietoturva

## **Preface**

I would like to thank my company and all the personnel that gave me the opportunity to do this research. I would also like to thank my supervisor Tommi Mikkonen for all their valuable feedback and guidance during this process.

And most of all i would like to thank my partner Susanna for being the biggest motivation and support, not just for this work, but for the whole five years of my university studies.

In Jyväskylä June 6, 2024

Aatu Laitinen

## Glossary

AST	Application Security Testing
DAST	Dynamic Security Analysis Tool
DevSecOps	Development, Security & Operations
DevOps	Development & Operations
CI/CD	Continuous Integration and Continuous Delivery/Deployment
Cloud computing	On-demand availability of computer system resources
IAST	Interactive Application Security Testing
IDE	Integrated Development Environment. Application for streamlining software development.
Microservices	Architecture where an application is built on multiple independent components.
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SDLC	Software Development Life Cycle
Version control	Practice of tracking and managing changes to code
YAML	Yet Another Markup Language. A human-readable data serialization language.

## List of Figures

Figure 1. The waterfall model of software development lifecycle (Adenowo and Adenowo 2013). .....	5
Figure 2. The agile model of software development lifecycle (Cois, Yankel, and Connell 2014). .....	6
Figure 3. Phases of continuous integration and delivery (Zhou et al. 2023). .....	7
Figure 4. DevOps lifecycle (Gunja 2023). .....	8
Figure 5. DevSecOps lifecycle (Marsal 2023). .....	9
Figure 6. Research workflow plan. ....	14
Figure 7. “Shades” of grey literature (Adams, Smart, and Huff 2017).....	15
Figure 8. Question 4 results. ....	33
Figure 9. The results of Q5, Q6, and Q7.....	34
Figure 10. The results of Q8, Q9, Q10, and Q11.....	35
Figure 11. The results of Q12, Q13, Q14, Q15, and Q16.....	37

## List of Tables

Table 1. White literature referenced in the literature review. ....	16
Table 2. Grey literature referenced in the literature review. ....	17

# Contents

1	INTRODUCTION .....	1
1.1	Need for agile security implementations through DevSecOps .....	1
1.2	Objective and research questions .....	2
1.3	Organization of the thesis .....	3
2	BACKGROUND .....	4
2.1	Software Development Lifecycle .....	4
2.2	Continuous software engineering .....	6
2.2.1	Continuous integration and delivery .....	6
2.2.2	DevOps .....	7
2.2.3	DevSecOps .....	9
2.3	Application Security Testing .....	10
3	RESEARCH DESIGN .....	13
3.1	Multivocal Narrative Literature review .....	13
3.2	Research process .....	14
3.3	Literature inclusion .....	15
4	LITERATURE REVIEW .....	18
4.1	Challenges affecting development efficiency in shift-left security .....	18
4.1.1	Challenges related to practices .....	18
4.1.2	Challenges related to people and collaboration .....	19
4.1.3	Challenges related to technology and tools .....	20
4.2	Solutions for ensuring the development efficiency .....	21
4.2.1	Solutions related to practices .....	21
4.2.2	Solutions related to collaboration .....	23
4.2.3	Solutions related to technologies and tools .....	23
5	CASE STUDY .....	25
5.1	Methodology and process .....	25
5.2	Dependabot configuration and usage .....	26
5.3	Questionnaire .....	28
5.4	Results .....	32
6	DISCUSSION .....	38
6.1	Implications of the literature review .....	38
6.2	Implications of the case study .....	39
6.3	Relevance to previous research on the subject .....	41
6.4	Future research .....	43
7	SUMMARY .....	45
	REFERENCES .....	46



APPENDICES .....	51
A Guideline for using Dependabot to automate dependency vulnerability fixing	51
A.1 Enabling Dependabot .....	51
A.2 Configurations for the security pull requests .....	53
A.3 Access to repository secrets needed for actions initiated by Depend- abot created pull requests.....	54
A.4 Assessing the automated security pull requests .....	55

# 1 Introduction

DevSecOps has risen as an answer to traditional security measures having hard time to adopt into DevOps (Rajapakse et al. 2022). Integrating security into DevOps processes is challenging in many aspects. One of these aspects is how DevSecOps can be adopted without affecting developers' ability to efficiently produce and deploy code, which is a fundamental part of DevOps. In a more standard DevOps process the security is usually taken into account after the initial deployment is done, this leads to instances where developers need to do constant redeploys to find and fix security vulnerabilities

A core component of DevSecOps is shift-left security which has been introduced to move the security processes from the end of the Software Development Life Cycle (SDLC) to the start (Anjaria and Kulkarni 2022). Shift-left security allows companies to detect and fix security issues in a much earlier state in the SDLC which leads to less security issues leaking into production and also reduces the need for developers to redeploy security fixes after the initial implementation. However, shifting the security processes to the left can lead to much more complicated and time-demanding development processes that affect the developers' ability to efficiently implement new code.

## 1.1 Need for agile security implementations through DevSecOps

The longer a security issue remains undetected or dealt with in the software development lifecycle, the more it costs to fix that issue since you need to repeat all the steps that it took for the features to get deployed in the first place (Planning 2002; Carter 2017). Catching the issues and dealing with them in the earliest parts of the development lifecycle can greatly reduce the costs of the development as testing usually accounts for a noticeable part of the overall resources used. But cost is not the only aspect that is negatively effected by late detection of security issues. If the issues escape into the production environment it means that the software is compromised, no matter how quickly the issue is then fixed (Sharrma 2021). This is why it is essential for modern software companies to switch to a more proactive way of catching the security issues as early as possible. When the emphasis on finding and fixing

security issues is shifted from the security teams to the developers, the pressure is released from security teams and their resources are freed up to be used for other purposes (Kuruville 2021).

One of the main issues that causes companies to overlook their security is the fact that it often comes at the cost of development speed (Mohan and Othmane 2016). The rapid speed at which software needs to be delivered in the modern market for software has raised the emphasis on agile methods of development such as DevOps. These methods rely on automation to achieve the speed and agility required for companies. Traditional manually executed security tools and practices have a hard time keeping up with this speed (Nawale 2022). This has led to the need to also automate the security tools and practices, so that they can be integrated into the agile process of development (Mohan and Othmane 2016). Automating the security tools also decreases the possibility of negligence towards performing the security tasks (Anjaria and Kulkarni 2022). But automating and integrating security into the earlier stages of agile development lifecycle without hindering the developers' efficiency is challenging and requires in depth research on how it can be achieved.

## **1.2 Objective and research questions**

In order to improve the industry's knowledge on how to maintain the development efficiency with DevSecOps, this work aims to identify challenges for maintaining effective development when shifting security to the left in the SDLC. On top of identifying core challenges to this problem, this work also reviews and combines different implementation that can be used to solve these challenges. This leads to the two research questions at the core of this work:

**RQ1 What challenges does shifting security left introduce to the development efficiency in DevOps?**

**RQ2 How can security implementations be shifted left without hindering the development efficiency?**

The first question focuses on different barriers to efficiency that developers face when the responsibility of performing the security practices is moved to be a part of the earliest stages of the development. To answer this question, this work aims to collect and categorize key

DevSecOps adaptation challenges that have been recognized in the industry. To develop an understanding on how these challenges can be dealt with, the second research question focuses on what practices and tools can be utilized to ensure that the development efficiency stays as high as possible when adopting the shift-left security principle. To provide an even more in depth look into one of these solutions, a case study is used to study an implementation for the solution.

### **1.3 Organization of the thesis**

This work combines academic and grey literature to map out what sort of challenges does left-shift security introduce to the developers' ability to efficiently produce software, and what solutions can be used to combat these challenges. In this work we also implement one of these solutions, the dependency vulnerability management with GitHub's Dependabot, to a real world software development environment and evaluate its serviceability.

The structure of this thesis is in the following manner. After this introduction chapter we orientate to the background of continuous software engineering and application security testing in chapter 2. In chapter 3 we describe the research process of this work as well as the research design of the literature review. Following this we conduct the literature review in chapter 4. In chapter 5 we conduct a case study and report its results. After that we discuss the results in more detail in chapter 6 as well as cover the implications of this work, discuss earlier research on the subject, and suggest future research. Chapter 7 provides a compact summary on the whole study.

## **2 Background**

This chapter contains background information about the nature of the software development lifecycle and how it has evolved to enable continuous software engineering. We also explain the concept of continuous integration and delivery and how its phases form a pipeline through which applications can be created and maintained. This chapter also explains the concept of DevOps and how the introduction of security evolves it to DevSecOps. We also cover a number of application security testing tools and practises that are discussed later on in this work.

### **2.1 Software Development Lifecycle**

The Software Development Lifecycle (SDLC) is a conceptual model that is used to represent the stages a software goes through during its development and operation (Cois, Yankel, and Connell 2014). An easy and traditional way to illustrate the software development lifecycle is through the "Waterfall Model" (Adenowo and Adenowo 2013). The waterfall model represents the stages of software development as a linear sequence where the development starts from the analysis of the requirements of the software and ends with the operation and maintenance of the software, as shown in figure 1.

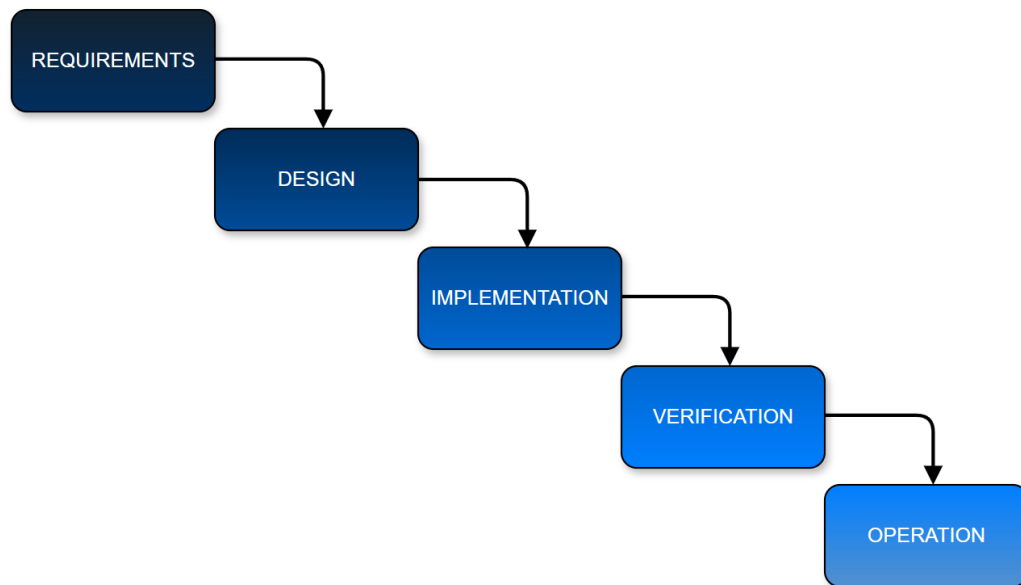


Figure 1. The waterfall model of software development lifecycle (Adenowo and Adenowo 2013).

The traditional waterfall model assumes that the requirements of the software are clear at the start of the development, and that no further changes to the requirements are introduced down the line of the lifecycle. The state of modern software development is far from this as new requirements and changes are introduced at a rapid pace, and new features need to be released as fast as possible (Cois, Yankel, and Connell 2014). This is why a more agile approaches have been taken to the software development lifecycle. These agile methodologies change the linear approach of the waterfall model into a circular iterative approach. This means that the software development lifecycle does not end in the operation and maintenance stage of the process, but goes right back to planning and requirement analysis as new requirements and changes are introduced for the software. The circular nature of the agile software development lifecycle is presented in figure 2.



Figure 2. The agile model of software development lifecycle (Cois, Yankel, and Connell 2014).

## 2.2 Continuous software engineering

This section explains how applications are created and maintained through continuous integration and delivery (CI/CD) pipelines. Here we also explain the concept of DevOps and how it forms around the SDLC and CI/CD Pipeline. In addition, we take a look at how the introduction of security into DevOps has evolved the concept into DevSecOps.

### 2.2.1 Continuous integration and delivery

A detailed way of defining the phases the application code takes during its SDLC is through Continuous integration and delivery (CI/CD) pipelines. CI/CD pipelines are a pivotal part in the continuous software development (Wessel et al. 2023). The continuous integration is a practice of automatically building and verifying all code that is created. Continuous delivery is the process of automatically deploying applications and providing feedback on them. (Soni 2016, pp. 28-29) Together these practices are used to form a pipeline through which applications are created. The CI/CD pipeline can be divided into five different phases

(Zhou et al. 2023). These phases are also used later in this work to showcase at which parts of the pipeline different security practices and tools take place. These five phases are in order: pre-commit, commit, acceptance, production, and operations. The pipeline is a continuous cycle that starts from the beginning every time a new change is to be made. The flow of the pipeline is illustrated in figure 3.

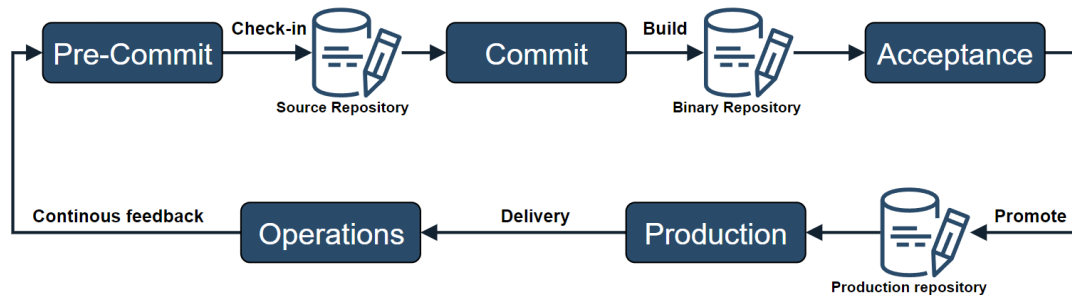


Figure 3. Phases of continuous integration and delivery (Zhou et al. 2023).

Pre-commit phase includes the development that is done on a developers machine before anything is actually committed to version control. Commit phase is where the code is build from the version control and transferred to a test environment. In the acceptance phase the code changes are validated in the test environment before they are deployed to a production environment. The deployment to production is the fourth phase in the pipeline. The final phase is the operation of the production environment that includes practices such as continuous monitoring. (Zhou et al. 2023)

### 2.2.2 DevOps

The field of software development has come to see an ever increasing growth in the speed and demand at which software needs to be created and delivered (Soni 2016, pp. 9-11). In the more traditional ways of working, the software development teams used to write code for fixes or new features and conduct testing for it before handing it over to the operations team for deployment and maintenance. In order to adapt to the speed of the modern field, DevOps emerged to integrate these two aspects into one cohesive unit (Ebert et al. 2016). DevOps



achieves this merge by introducing agile methods such as build automation, continuous integration and delivery, as well as continuous monitoring and logging of the applications' conditions. However, these methods alone can not enable the shift to efficient DevOps adaptation. A cultural shift is needed for the whole company that promotes the DevOps ways of working. Traditional software architecture also makes DevOps adaptation a challenging feat to achieve. This is why companies have started to shift from more traditional architecture models to modern approaches such as microservice architecture and cloud based solutions.

The DevOps lifecycle can be divided into eight different stages that are continuously iterated. In the software development lifecycle the development part contains the planning, coding, building and testing of the software, and the operations part includes the release, deployment, operation, and monitoring of the software. DevOps merges these parts into a continuous cycle as presented in figure 4. The DevOps process is continuously monitored and evaluated through out all parts of the cycle. Improvements are made continuously to the process based on the feedback received by monitoring it. (Gunja 2023)

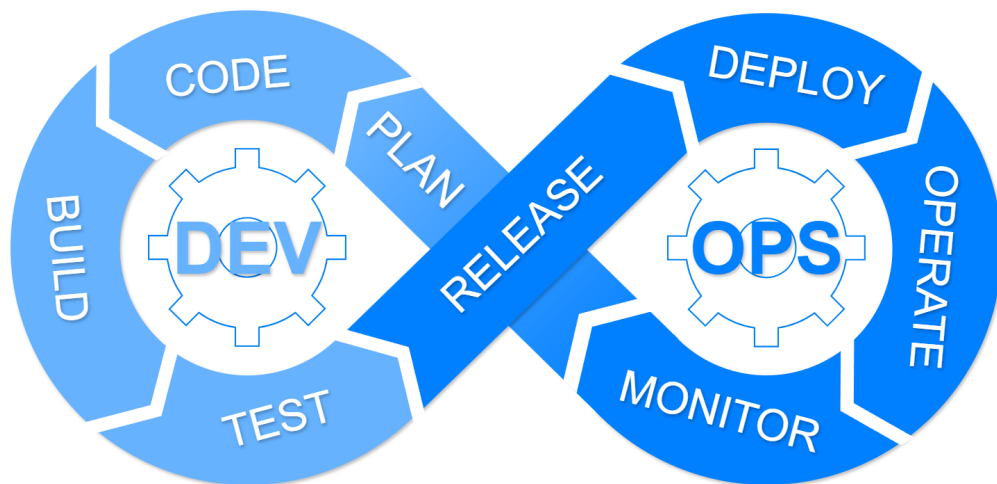


Figure 4. DevOps lifecycle (Gunja 2023).

### 2.2.3 DevSecOps

To answer to the security challenges in DevOps caused by the inability of traditional security methods to be able to keep up with the speed, a new evolution of the DevOps methodology emerged; DevSecOps (Myrbakken and Colomo-Palacios 2017). DevSecOps aims to seamlessly integrate security into the two core disciplines of DevOps, as visualized in figure 5. Similar to DevOps, DevSecOps relies on the combination of agile practices and techniques along with comprehensive cultural commitment to achieve its goals. DevSecOps goals are to both increase the security of the software as well as increase the efficiency at which security can be implemented. (Marsal 2023)

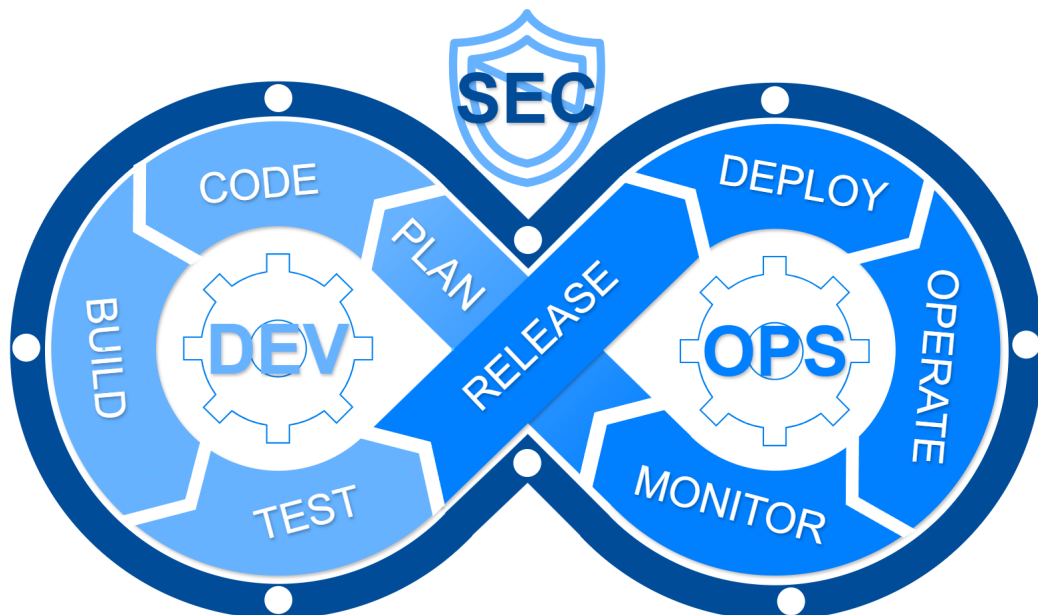


Figure 5. DevSecOps lifecycle (Marsal 2023).

A core component for successfully implementing DevSecOps is to shift-left the security. Shifting security left means assessing security issues in the earlier stages of the SDLC. Another core component is the automation of the security practices to ensure that the security compliance is maintained at all times. In order to succeed in the implementation of DevSecOps, continuous collaboration and training is required between the development, operation,

and security teams. If the whole organization is not embracing the DevSecOps principles, its implementation is likely to fail. (Marsal 2023)

Practices that enable DevSecOps can be divided into two different aspects that work in unison to make achieving DevSecOps goals possible (Marsal 2023). The first aspect is technological enablers such as security infrastructure and security practices that can be automated and integrated to be part of the SDLC. The second aspect is cultural enablers such as company wide security awareness, efficient communication, and responsibility sharing. The technology related practices can further down be divided into practices that either harden the DevSecOps processes or speed them up (Zhou et al. 2023).

### 2.3 Application Security Testing

There are countless different tools and practices that can be used to improve an application's security. This section provides background on commonly used Application Security Testing (AST) tools and practices that are relevant to this study.

**STRIDE** is a threat model that is used to identify and categorize different types of threats on a software application. STRIDE is an acronym for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of Privilege. The goal of STRIDE is to identify what attackers are trying to achieve. When the type of the threat is identified, correct techniques can be utilized to prevent them from happening or to mitigate their effects in case they occur. (Conklin 2024)

**Software Composition Analysis (SCA)** is a security method that can be used to give visibility into the open source software usage of an application and to track the possible security issues that they introduce to the application. It is ever more common for companies to utilize external open source components when building their application. SCA helps with managing with the security risk of using these components. Many of these open source software components also use other open source software components which can lead to the creation of long supply chains. Manual tracking of the security issues in all these components has grown to be very insufficient. SCA tools can be used to track the whole component supply chain and to detect security and licensing issues at every step of this chain. SCA can be

used to detect possible issues when deploying new code as well as to continuously monitor deployed applications for appearance of new issues and vulnerabilities. (“What is Software Composition Analysis?”, no date)

**Static Application Security Testing (SAST)** is a method that can be used to analyze source code to find security-related issues and vulnerabilities. SAST tools can be added as an extension into a Integrated Development Environment (IDE) which allows developers to get real-time feedback when writing new code. This makes it possible for developers to detect and fix new security issues even before committing their changes to version control. On top of pointing out possible security issues, SAST tools can also help visualize the problems through graphical representations and give detailed guidance on best practices for fixing the issues. This makes it easier for developers to fix these security issues without prior expertise on them. One of the key advantages of SAST tools is that they do not require the application they are testing to be deployed. Since SAST tools analyze the source code without the need to execute the application, they can be used on a wide variety of different applications. SAST promotes white box testing, which means that the developer can see the root cause of the problem as the issues appear, making it much faster and cheaper to fix these issues. (“Static Application Security Testing”, no date)

**Dynamic Application Security Testing (DAST)** is a testing methodology that is used to test security vulnerabilities that appear at the runtime of the application. DAST is a "black box" testing methodology meaning that the DAST tools do not have inside access into the applications source code. DAST tools are used to perform series of malicious inputs and attacks in order to try to penetrate the security of the application, similar to what a real attacker would try to do. DAST tools provide a comprehensive and cost-efficient way to use automated scans that simulate the real-world environment of cyberattacks. (Bashvitz 2023)

**Interactive application security testing (IAST)** is a method to test application security while the application is being run. The testing can be done either by a human tester or an automated unit that interacts with the application. IAST tools consists of sensor modules that are methods in software libraries that have been implemented into the software code to track the software’s behaviour while certain things are being executed. The sensors produce real-time feedback on the software’s behaviour. IAST tools are costly to implement as

they require the sensor modules to be implemented into the software. IAST tools extend the functionality of DAST tools in such a way that allows the tools to also see how the the application behaves internally, not just externally. They also extend the functionality of SAST tools by taking into account various scenarios that only rise during the runtime. (“Interactive Application Security Testing” 2022)

**Dependency management bots** have in the recent times seen a noticeable increase in their use among developers as a way to automate the process of updating dependencies (Wu et al. 2022). In the context of application security, dependency management bots are used to automatically detect vulnerable dependencies and create pull requests that update those dependencies (He et al. 2023). These bots are useful at increasing the developers awareness on possible dependency issues and to speed up the process of fixing them. GitHub remains as the most widely used social development platform (Wessel et al. 2023). This has also reflected on the adaption rate of it’s build in dependency management bot, Dependabot.

## 3 Research design

In this chapter we provide reasoning and background on the research methods that have been chosen for conducting the literature review part of this study. This chapter also covers the research process used to conduct this study. The included literature is also documented here along with their inclusion criteria.

### 3.1 Multivocal Narrative Literature review

Narrative (also known as traditional) literature review is a research method that can be used to used to summarize and critique a collection of literature surrounding a subject area (Cronin, Ryan, and Coughlan 2008). It is used in this work to provide a comprehensive summary of the state of DevSecOps adaptation challenges affecting development speed as well as publicly recognized or suggested solutions to these challenges. The goal of this literature review is to create an understanding of the current knowledge in the subject area and to provide comprehensive background for at least one practical solution that will be tested and evaluated later in this work.

Since the subject of DevSecOps is still relatively new there is only a limited amount of academic peer-reviewed literature, also referred to as white literature, available on the subject. To compensate the lack of this academic literature, grey literature is also included into the narrative literature review, making it also a multivocal review. A multivocal literature review includes all accessible writings of a topic into the literature review, creating a more diverse view on a matter as there are many different types of voices from which the review is structured from (Garousi, Felderer, and Mäntylä 2019). Grey literature comes in a variety of forms that embody different perspectives and purposes on the topic in question. According to Garousi's, Felderer's, and Mäntylä's study (2019) the use of grey literature should not be avoided, but on the contrary is recommended for research done in the field of software engineering. But for the research to accurately reflect the current situation, criteria should be set in place for which grey literature is to be included or excluded.

## 3.2 Research process

The research process that is followed in this work is split to five main steps and one continuous step that is iterated through out the first four steps of the process. After reviewing and researching previous works related to this works subject, a multivocal narrative literature review is conducted in two parts. The first part of the literature review focuses on finding challenges related to the adaptation of DevSecOps from the development efficiency view-point. The second part of literature review focuses on researching possible solutions to the problems found in the first part. After the second part of the literature review, a case study is conducted on one of the identified solution. Research and documentation on related background information is continuously done through all these steps. After these steps the results are summarized and further discussed. The complete workflow of the process is presented in figure 6.

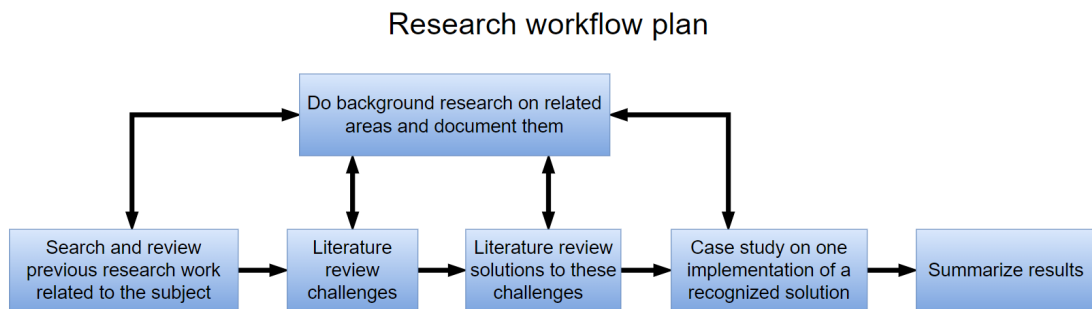


Figure 6. Research workflow plan.

As is the standard way to conduct a narrative literature review, this research is performed by using selected keywords to identify important literature (Cronin, Ryan, and Coughlan 2008). The primary keywords used to search literature in this research are "DevSecOps", "Shift-left", "security", and "challenges". These keywords are seen as the most describing of the main topics of this work. To further narrow down the literature, additional keywords such as "DevOps", "development", "efficiency", and "solutions" are used. Some aspects of the research might require more detailed keywords, such as names of specific security tools and practices. In those cases these names are used as keywords. Nested references to other

potentially useful pieces of work are also used when found in the literature selected through the use of the keywords. The search engines that are used for finding academic literature with these keywords are Google Scholar and JYKDOK online library.

Grey literature used in this work is judged and selected using the "shades of grey" scale by Adams, Smart, and Huff (2017). The scale is visualized in more detail in the figure 7. When selecting literature that has limited control of outlet and source expertise in software engineering research, it is important to identify the type of producer behind the grey literature (Garousi, Felderer, and Mäntylä 2019). To ensure the quality of this literature review, only grey literature where the person or organization acting as the author of the literature can be recognized as a legitimate expert of the field will be used. This means that the grey literature included in this work will be 1<sup>st</sup> and 2<sup>nd</sup> tier grey literature. 3<sup>rd</sup> tier literature will not be included in this work as the expertise of their source is most likely too difficult to determine. Google search engine is used for finding the grey literature that is used in this work.

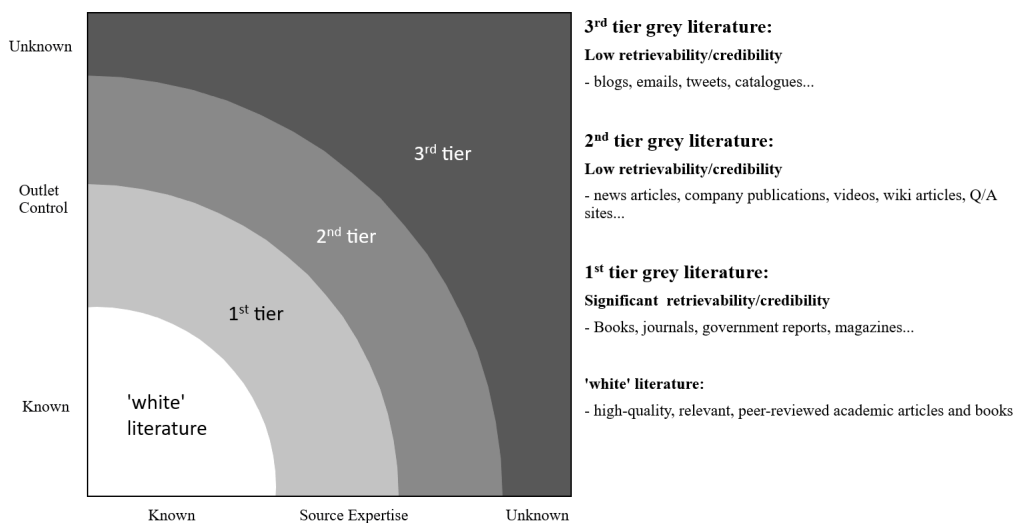


Figure 7. “Shades” of grey literature (Adams, Smart, and Huff 2017).

### 3.3 Literature inclusion

The total amount of different literature that was referenced in the literature review part of this work added up to 23 pieces. Out of these pieces of literature, 14 pieces were academic peer-



reviewed literature and the remaining ten pieces were grey literature. The most referenced piece of literature was the Rajapakse’s et al. (2022) article on challenges and solutions with DevSecOps adaptation. This article saw 12 references with eight in the section focusing on challenges and four in the solutions section. This articles influence on the literature research is noticeable as the structure at which the challenges are categorized is adapted from the structure used in Rajapakse’s et al. work. Other pieces referenced in the literature review have been used at most once or twice, and they are in most cases used primarily to deepen the knowledge around the subjects introduced through Rajapakse’s et al. article. All 14 referenced pieces of academic literature and the context in which they were referenced are shown in table 1

Table 1. White literature referenced in the literature review.

<b>n.</b>	<b>Referenced literature</b>	<b>Number of references</b>	<b>References for challenges</b>	<b>References for solutions</b>
1	Rajapakse et al. 2022	12	8	4
2	He et al. 2023	2	1	1
3	Koskinen 2019	2	2	
4	Agaronian 2021	1		1
5	Alfadel et al. 2021	1		1
6	Colliander 2022	1	1	
7	Decan, Mens, and Constantinou 2018	1	1	
8	Gonzalez, Perez, and Mirakhorli 2021	1	1	
9	Hilton et al. 2017	1		1
10	<i>JFrog Curation Redefines “Shift Left” Security for Enterprise Software Supply Chains</i> 2023	1	1	
11	Kumar and Goyal 2020	1		1
12	Morales et al. 2020	1		1
13	Myrbakken and Colomo-Palacios 2017	1	1	
14	Zhou et al. 2023	1		1

In addition to the white literature, ten pieces of grey literature were used to widen the view on the matter by including practitioners’ insights from outside of academic literature. The inclusion criteria for the grey literature was that the literature could be categorized under

the 1<sup>st</sup> or 2<sup>nd</sup> tier in the "shades of grey" scale. All of the grey literature excluding the Synopsis (2023) report were 2<sup>nd</sup> tier literature, where the outlet was a publicly recognizable and creditable software company or institution. All the references to grey literature are listed below in table 2

Table 2. Grey literature referenced in the literature review.

<b>n.</b>	<b>Referenced literature</b>	<b>Number of references</b>	<b>References for challenges</b>	<b>References for solutions</b>	<b>Tier</b>
2	www.castsoftware.com, no date	1		1	2 <sup>nd</sup>
3	Nawale 2022	2	2		2 <sup>nd</sup>
4	Nugraha 2024	1		1	2 <sup>nd</sup>
5	Patni 2024	1	1		2 <sup>nd</sup>
6	“5 benefits of shift left security” 2021	1	1		2 <sup>nd</sup>
7	“Guide to software composition analysis (SCA)” 2022	1		1	2 <sup>nd</sup>
8	“DevSecOps and Shift Left Security: A Guide”, no date	1		1	2 <sup>nd</sup>
9	<i>State of DevSecOps in Europe</i> 2023	1	1		1 <sup>st</sup>
10	“Static Application Security Testing”, no date	1		1	2 <sup>nd</sup>

## **4 Literature review**

In this chapter a multivocal narrative literature review was conducted to identify different challenges related to maintaining development efficiency when adopting shift-left security into DevOps. We also reviewed what kind of solutions can be used to answer these challenges.

### **4.1 Challenges affecting development efficiency in shift-left security**

Adaptation of DevSecOps and shift-left security arises a wide arrange of challenges with different natures. This section focuses on those challenges and how they affect developers and companies in regards of maintaining their development efficiency. In their qualitative analysis on these challenges, Gonzalez, Perez, and Mirakhorli (2021) identified seven different types that are a frustrating for developers to deal with. On the other hand, Rajapakse, Zahedi, Babar, and Shen (2022) categorized different DevSecOps adaptation challenges into four different themes; tools, practices, infrastructure, and people. Based on these studies, the challenges that are connected to maintaining the development speed and efficiency can be divided into three primary themes. Those themes are challenges related to practices, challenges related to people and collaboration, and challenges related to technologies and tools.

#### **4.1.1 Challenges related to practices**

Most of the traditional security practices are time consuming and require to be manually performed, which makes them challenging to implement into a DevOps pipeline. The DevOps pipelines are formed mostly around continuous and automated practices to enable faster releases of new implementations. One of the biggest challenges emerges here as the integration of security requires a way to find around the need of human input that is present in the traditional security practices (Patni 2024). The abilities to fully automate these traditional security practices have been very limited (Rajapakse et al. 2022). This means that the emphasize has been more on finding and developing new types of security activities, that can be automated and can keep up with the rapid development cycles of DevOps (Koskinen 2019).

To make things even more challenging for incorporating the security into the DevOps pipeline, it is usually not enough that the security can match the fast deliveries. It is crucial for companies that development can also retain its high quality alongside the fast delivery speed. (Koskinen 2019) This means that security practices also need to support the maintaining of the development quality.

With the vast majority of code in today's software products being from third-party libraries and frameworks, the amount of vulnerabilities introduced through them has increased drastically (Decan, Mens, and Constantinou 2018). When security vulnerabilities are discovered in those third-party components they are often also quickly patched. But for other products that are using these components, they also need to update the components to the latest patches that include these security fixes in order to avoid the vulnerabilities from spreading into their products. Updating these components can quickly become a very demanding task as more and more libraries are used. When the security responsibility of performing these updates is shifted to the developers it can drastically hinder their ability to efficiently produce valuable software (He et al. 2023). When new vulnerabilities are constantly discovered and fixed, a lot of time has to be invested by developers to make sure the libraries they use are up to date and include these security fixes.

#### **4.1.2 Challenges related to people and collaboration**

When adopting a DevSecOps culture, one of the key aspects for its success are the people and their collaborations toward the common goal of efficient DevSecOps. It is typical for software development and operation teams to work in "silos", meaning that they work mostly isolated from other teams. This means that the communication and collaboration between the different teams and stakeholders is inefficient (Rajapakse et al. 2022). When the security teams are introduced into the picture, the inter-team collaboration gets even more complicated. Because the developers usually lack the security expertise of a security team, the need for collaboration between the development and security teams increases even more (Nawale 2022).

The security teams are primarily the most concerned with how secure the development pro-

cess and its outcomes are. The development teams however focus more on how the development can be done effortlessly and as fast as possible. These conflicting interests can further challenge the teams' ability to collaborate (Rajapakse et al. 2022; Nawale 2022). Further friction between the teams can arise as developers and operators might see the security team's demands as a hindrance to their workflow and development speed (Myrbakken and Colomo-Palacios 2017).

### **4.1.3 Challenges related to technology and tools**

Much like the traditional security practices, traditional security tools often have a hard time adapting into the DevOps development lifecycle and are usually unable to be automated (Rajapakse et al. 2022). According to a research by Censuswide and Synopsys Cybersecurity Research Center, AST tools' inability to fit in to the rapid release cycles of DevOps is seen as a major issue in the industry (*State of DevSecOps in Europe* 2023). This inability may lead to development teams completely abandoning the use of these security tools. Another reason that prevents teams from integrating these tools to their DevOps pipelines is that the initial work required for them to be included is often challenging and time consuming. (Rajapakse et al. 2022)

The use of open-source software components has become a stable way for developers to dramatically increase their delivery speed (*JFrog Curation Redefines "Shift Left" Security for Enterprise Software Supply Chains* 2023). However the use of these components also heavily increases the risk of introducing malicious vulnerabilities through the third-party components. With the ever increasing usage of third-party components, the complexity of catching all the possible vulnerabilities introduced through the dependencies of these components has increased the difficulty of security assurance by a large margin (Rajapakse et al. 2022). This complexity has also led to the decrease of speed at which these components can be safely used to deliver new code. This has raised the need for tools that can help with handling these complexities.

Proper configuration of security tools is also a major challenge in DevSecOps. Poor configurations can lead to the tools reporting lots of false positive issues. SAST tools in particular

are very prone to false findings when not correctly configured. (Colliander 2022) Spending time to triage or even unnecessarily fix these false positives can cause a lot of additional work for the developers and thus decrease the development speed. It is also important that the tools are able to determine the severity of the defects they find. The developers need information about what defects are trivial and what are crucial for the security, so that time is not wasted on fixing the less important issues.

SAST tools integrate well to the DevSecOps as they require no deployment of the application or execution of the code and as a result can be run at a very early state in the SDLC. The downside to SAST tools however is that they cannot detect issues and vulnerabilities that arise on the application runtime. This is where DAST tools come in handy as they provide analysis and detection on security issues that can arise during the runtime (“5 benefits of shift left security” 2021). However DAST tools take much longer to run and require more manual effort which results with them having hard time with adapting to the fast and frequent release cycles in DevOps (Rajapakse et al. 2022).

## **4.2 Solutions for ensuring the development efficiency**

In this section we review suggested solutions to the challenges that were recognized in the previous section. The solutions are categorized based on whether they are related to practices, people and collaboration, or technologies and tools.

### **4.2.1 Solutions related to practices**

Developers should stay cautious to only create security tests that contribute meaningful value for the software. It is also important for development teams to allocate time for maintaining their test suites and to clean up leftover tests that are no longer valuable. All tests require resources every time they are run in the CI/CD pipeline, so it is important to minimize the amount of unnecessary tests and prevent them from over flooding the build process. This may not have a large impact on a singular deployment and the allocation of resources towards cleaning up the testing pipeline may have temporary setback on development speed. But overtime this reduced build time helps the CI/CD pipeline to stay efficient. (Hilton et

al. 2017)

Threat analysis practices such as STRIDE can be used to detect and understand security threats at different parts of the CI/CD pipeline (Rajapakse et al. 2022). STRIDE is traditionally a model manually used by security experts. However, various DevOps practices can be used to automate the STRIDE model to be a part of the development pipeline, and thus decreasing the manual work required for its usage (Nugraha 2024).

In order to ensure that the shift-left security is successfully adapted, it is important to also measure its implementation (“DevSecOps and Shift Left Security: A Guide”, no date). Two useful ways to measure how the shift-left security affects the development speed is to measure how much time is used on manual security reviews, and to measure the percentage of security requirements that have been automated into the development pipeline. The larger the percentage of automated security requirements the faster the developers’ ability to receive feedback about them. This also usually leads to less time spent on manual security reviews which then translates to more time used on the actual development.

To ease up the massive load of having to constantly update third-party libraries to ensure the latest security fixes are in place, the practice of using dependency management bots has become increasingly more common (Agaronian 2021; Alfadel et al. 2021). These dependency management bots such as GitHub’s Dependabot are used to detect outdated and insecure dependencies and to automatically create a pull request with an update (He et al. 2023). These bots alleviate a lot of manual work from developers such as constantly checking dependency monitoring solutions for dependency issue alerts and creating and documenting update pull request for these issues. However, adapting the usage of these automated pull requests is not all easy and simple. Developers are often suspicious about if the changes proposed by the dependency management bots will break their code, which makes them hesitant about merging those changes. Also, if the bots are constantly adding more and more pull requests, developers might quickly become overwhelmed and annoyed by them. This could lead to developers ignoring the pull requests. In order to avoid these types of scenarios, a lot of thought needs to be put into the configuration of these bots. Configuring for example the severity of the issues that are considered for the automated pull requests to be more visible or using grouped pull requests can reduce the frustration and effort that developers could face when

resolving these pull requests.

#### **4.2.2 Solutions related to collaboration**

When the development, operation, and security teams work in isolation and the communication and collaboration infrastructure is not properly build around mutual processes and technologies, adopting a fast based collaboration environment becomes difficult (Kumar and Goyal 2020). Creating a concept of a continuous security framework that defines common practices and technologies for how to collaborate efficiently between the teams can mitigate these difficulties. Favoring multidisciplinary teams, having clear and standardized communication practices that enable short feedback cycles, and increasing the involvement of developers in security related assignments are among some of the practices that can be used to increase the efficiency of inter-team collaboration (Rajapakse et al. 2022).

One possible solution for addressing the friction between development and security teams and to compensate for the lack of security expertise of individual developers is to assign security champions for the development teams (Rajapakse et al. 2022). Security champion is a developer, usually the one with the highest knowledge in security, that is chosen to act as the point of contact between the security team and the development team (“Security Champions” 2022). The security champion can streamline the communication between the security and the development teams by being up to date with the state of both sides. Security champions can help the development teams with integrating the security practices into their development process, and to provide feedback about integration issues to the security team. This can reduce the inconvenience that security practices might cause to developers as the hindrances to development efficiency are more likely to be adjusted to when developing the security practices.

#### **4.2.3 Solutions related to technologies and tools**

To solve the challenges regarding the use of SAST and DAST tools in DevOps environment, Interactive application security testing tools can be used. IAST is a modern class of security testing that combines the features of SAST and DAST tools into a set of security tools that



can be more successfully integrated as an automated part of the fast paced CI/CD pipeline. (Rajapakse et al. 2022)

Dependency scanning tools such as SCA tools have been widely taken into use to combat the complexity and vulnerability of using large amounts of third-party software components (www.castsoftware.com, no date). SCA tools also fit into the DevSecOps pipeline since they can be automated to be run every time new features are pushed forward at different parts of the SDLC (“Guide to software composition analysis (SCA)” 2022). However, relying solely on SCA to catch security issues when new features are pushed to version control can become time consuming for developers. This would be so because the developers would need to push their implementations to the point in the CI/CD pipeline where the SCA scan are run in order to see if they have security issues.

Fortunately, just like SAST tools, SCA scanning tools can be shifted as far left as to be run in the developer’s integrated development environment (IDE) in the pre-commit stage of the CI/CD (Morales et al. 2020). This means that developers can catch and fix dependency and source code related vulnerabilities as they are writing new code. But relying solely on developers to always run these scans and to fix these issues is insecure which is why the IDE scanning should be paired with the automated scans in the pipeline. SCA and SAST scans can be configured so that they are run every time in the commit phase of CI/CD pipeline (Zhou et al. 2023). This way most of the vulnerabilities can efficiently be dealt with as early as possible, but in the case that the developers are unable to catch the defects in the pre-commit stage, the automated SCA and SAST scans in the pipeline ensure that no insecure code goes forward to the production.

## 5 Case Study

In this chapter a case study will be conducted. The study focuses on the use of dependency management bots for automated dependency fixing which was recognized as one of the solutions for maintaining development efficiency in the previous chapter. The solution will be studied and implemented into a real life company environment. A medium size Finnish software company will act as the testing environment. A questionnaire is used to study the perceived effects of the implementation among the developers of the company. The goals of the case study are to determine if the use of dependency management bots can make the dependency vulnerability fixing process more efficient, how this can be achieved in practice, and what barriers and enablers there exists for the success of the implementation.

### 5.1 Methodology and process

In this case study we take a more in depth look at how the GitHub's Dependabot can be adopted in order to maintain development teams' efficiency by automating the task of fixing dependency vulnerabilities through automated pull request (GitHub 2024b). The Dependabot is configured and adopted into the development environment of the development teams in a Finnish software company. Literature on the subject is studied to determine the best potential configuration for achieving the largest benefit for efficient development from the Dependabot. The reason why Dependabot was chosen for this case study among all the identified DevSecOps solutions was because it was seen as the most impactful solution by the target company's personnel.

This case study is conducted in five parts. The first part is to determine and create a configuration for the Dependabot implementation. The second part is to create a guideline for developers on how to enable the Dependabot into their CI/CD pipelines and how it's usage process works. The third part is to create a questionnaire that gathers feedback on the perceived usefulness of the adaption of Dependabot's automated security pull requests as well as about complications in its implementation. The fourth step is to hold demonstration sessions of the Dependabot implementation for the development teams and to distribute

the questionnaire to the team members after. The final step is to analyze the results of the questionnaire.

## 5.2 Dependabot configuration and usage

Dependabot documentation states that the automated security updates' behaviour can be configured using a `dependabot.yml` configuration file (GitHub 2024a). However in this case study we were unable to add the desired configurations through the `dependabot.yml` file. A workaround to this problem was to use GitHub Action that can detect when an automated Dependabot security fix pull request is created and modify it by adding extra labels and assigning reviewers for it. The code for the GitHub Action YAML that was created to perform these tasks is displayed below in listing 5.1.

Listing 5.1. `configure_dependabot_security_update.yml`

```
name: Label Dependabot security updates and assign reviewers
on:
  pull_request_target:
    types: [opened, synchronize, reopened, labeled, unlabeled]

jobs:
  handle-dependabot-security-updates:
    runs-on: ubuntu-latest
    steps:
      - name: Check if PR is an automated dependabot security update
        run: |
          title="{{ github.event.pull_request.title }}"
          labels="{{ github.event.label.name }}"

          if [[ "$labels" == "dependencies" && "$title" == "Bump"* ]]; then
            echo "SECURITY_UPDATE=true" >> $GITHUB_ENV
          else
            echo "SECURITY_UPDATE=false" >> $GITHUB_ENV
          fi

      - name: Add label
        if: env.SECURITY_UPDATE == 'true'
        uses: actions/github-script@v6
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          script: |
            github.rest.issues.addLabels({
```

```

        owner: context.repo.owner,
        repo: context.repo.repo,
        issue_number: context.issue.number,
        labels: ['Dependabot Security Update']
    })
- name: Assign Reviewer
  if: env.SECURITY_UPDATE == 'true'
  uses: actions/github-script@v6
  with:
    github-token: ${{ secrets.GITHUB_TOKEN }}
    script: |
      github.rest.pulls.requestReviewers({
        owner: context.repo.owner,
        repo: context.repo.repo,
        pull_number: context.issue.number,
        reviewers: [
          '<github-username1>',
          '<github-username2>',
        ],
      })

```

This action detects the Dependabot automated pull request by checking that it has the dependencies label and the pull request starts with "Bump" keyword, which are both apparent in all automated security updates created by Dependabot as of the time of writing. The action then adds a custom label that is used to better indicate what type of pull request it is in the pull request listing. The action also assigns reviewers to that pull request based on the GitHub usernames assigned in the GitHub Action, as shown above in listing 5.1.

Sets of repositories are usually assigned to development teams. The teams should determine which developers are responsible for which repositories, so those members' GitHub accounts can be configured to be automatically assigned as the reviewers for all automated pull requests created by Dependabot for the repository that they are responsible for. This assures that there is always at least some developers that are notified when a new vulnerability is detected and a fix is created by Dependabot. Being specifically assigned as a reviewer might also increase the likelihood of the developer actually reviewing the pull request.

As a part of the case study, a guideline was created for developers that shows in detail how Dependabot can be used and configured to enable automatic package vulnerability fixing for their repositories. The guideline is presented in appendix A. To ensure that all the developers

that partake in the questionnaire have a basic level of understanding of how the Dependabot security pull requests and their resolving process works, the process was demonstrated to all the development teams in addition to providing the guideline. The demonstrations were held in an online video call setting.

### **5.3 Questionnaire**

To achieve the goals of this case study, a questionnaire is used to collect developers insight and prejudice on the usage of GitHub's Dependabot with the guidelines and configurations presented in this work. An effective and widely used measurement for evaluating subjects opinions, attitudes, and feelings toward specific subjects is the Likert-scale (Nemoto and Beglar 2014). In this case study the Likert-scale is used to determine the level of agreement the developers have for different aspects regarding to the usefulness of the Dependabot as well as the obstacles surrounding its usage.

The content of the questionnaire is based on the existing literature on the matter as well as hypotheses created in this work. The questionnaire is constructed loosely following the five step process of Likert-scale questionnaire development (Nemoto and Beglar 2014). The first step is to create an understanding around the construct. Second step is to form the questionnaire items based on that understanding. The third step is to determine the range of the scale at which the items are answered to. The fourth step is to determine how the responses are measured and analyzed and the final step is to consult others about questionnaire and piloting it before distributing it to the target audience.

Since the target audience of this case study is small, a 6-point Likert-scale is used. This is because it is more accurate at capturing respondents overall sentiment compared to a lower point scale when a large target audience is not available and because studies have shown that it is not advisable to use a Likert-scale larger than six ("Everything You Need to Know About the Likert Scale in 2024" 2024; Nemoto and Beglar 2014). The scale ranges from strongly disagree (1) to Strongly agree (6) or equivalent if the question is presented in a way that requires a different wording for the options. As for the platform for conducting this questionnaire, Google Forms was chosen as it is already commonly used within the target com-

pany. Google Forms is also an efficient tool for this purpose since they are easily distributed and require no one-to-one interaction between the form distributor and the respondent. The questionnaire is maintained at short enough length so that developers have time to respond to it between their other daily tasks and don't shy away from it (Shull, Singer, and Sjøberg 2007, pp. 15-16).

In the questionnaire we research whether the usage of Dependabot security updates has a possible impact on development speed and increase of security. We also research at what level do developers trust the automated pull request and how can it be affected. We also research how eager developers are at utilizing the automated security updates. The questionnaire is divided into five different sections. The first section focuses on gathering basic information about the participants background and previous experience with the subject to give deeper context for the results. The questions about the background of the developer are as follows:

Q1 What is your job title?

Q2 Do you have any experience with GitHub Dependabot and its automated security or dependency update pull requests?

Q3 Do you have any experience with any other tool and its automated security or dependency update pull requests?

Q4 how often do you need to update dependencies in your projects?

The next three sections of questions contain the main questions of the questionnaire. All three of these sections only include Likert-scale questions that measure the developments agreement with the statement presented in the question. The usage of dependency management bots has been identified as a way to speed up the dependency update process, as well as to increase the overall security and awareness to vulnerabilities in software component supply chain (Agaronian 2021, Alfadel et al. 2021; He et al. 2023). This section includes questions that aim to gather the participants perception about how useful the Dependabot can be for them in terms of improving their projects' security and increasing their development efficiency. The questions in this section are as follows:

Q5 Automated security pull requests increase your awareness on possible vulnerabilities

in your projects dependencies.

Q6 Automated security pull requests make fixing dependency vulnerabilities faster for you.

Q7 Automated security pull requests improve the overall security of your projects.

Update suspicion was recognized as one of the key barriers to utilizing the dependency management bots (Agaronian 2021; He et al. 2023). This means that developers are worried that the changes introduced by the automated pull requests might break something in their projects and thus do not trust them enough to push them forward even if they have a low chance of breaking anything. Alfadel et al (2021) go as far as to suggest using automated merging for the pull requests which would require an even higher level of trust from the developers, but would also significantly increase the handling time of the vulnerability fixes. In this work we hypothesize that the unfamiliarity with the source code has a noticeable effect on how much the developers trust the introduced fixes. The next section has questions that focus on these trust related issues that the developers may face with the usage of Dependabot and factors that might reduce these issues. The questions in this section are:

Q8 You are worried that the automated security pull requests will break your code (and cause you additional work because of it).

Q9 You think that reviewing the automated dependency updates to the extent that you trust them (to not break anything) can be done effortlessly.

Q10 You would trust the Dependabot to merge pull requests automatically without human review.

Q11 You are more likely to avoid reviewing an automated security pull request if you are not familiar with the code it is updating.

In order for the automated vulnerability fix pull requests to consequentially speed up the development pipeline, the pull requests need to be actually acted upon and not just be ignored by the developers. Agaronian's (2021) study reveals that a large portion of the automated vulnerability fixes are still done manually despite Dependabot providing pull requests for them. Alfadel's et al. (2021) study also found several factors that lead to the automated security pull requests to not be merged. In this study we hypothesize that being specifically

assigned to review an automated pull request increases the likelihood of assessing it. One key concern for the pull request negligence was notification fatigue (He et al. 2023). Large amount of automated pull requests can feel overwhelming for developers to deal with which is why Dependabot's ability to group multiple vulnerability fixes to a single pull request is promoted. Severity of the vulnerability was also identified as a meaningful factor for how long an automated vulnerability fix remains unaddressed (Agaronian 2021). This last main section of the questionnaire gathers insight on these aspects surrounding the negligence of automated security updates and how developers can be encouraged to have a faster response to the pull requests. The following questions are included in this section:

- Q12 You are more likely to verify the automated pull request if the responsibility has been specifically assigned to you.
- Q13 Large amount of dependency vulnerability pull requests can feel overwhelming and discourage me from reviewing these pull requests.
- Q14 You would rather review a single pull request that fixes multiple dependency issues than to review multiple pull request that each cover a single fix.
- Q15 You are less likely to ignore an automated pull request if the vulnerability it fixes is of higher severity.
- Q16 You are more likely to assess an automated pull request faster if the vulnerability it fixes is of higher severity.

In order to be able to get unique insights on the subjects touched in the previous sections, an optional section is included at the end of the questionnaire. This section includes a question estimating the work demand of the pull requests to further reflect on the required efforts of reviewing and releasing the automatically created pull requests, as touched in question nine. The options for this are adapted from the findings on Dependabot security pull request merging and closing times in Alfadel's et al (2021) study. This section also has two open form questions where developers can share how they prefer to be notified about new security pull request, and if they have some other worries about the automated pull requests that were not covered in the previous questions. These questions are:

- Q17 How long do you estimate that it would take you on average to check, review, and



release an automated pull request? (given, it doesn't require breaking changes that need extra work)

Q18 What do you see as the best way to get informed about new pull requests created by Dependabot?

Q19 What kind of circumstances or issues do you think would most likely drive you to avoid reviewing the automated security pull request?

## 5.4 Results

The questionnaire yielded a total of 27 responses from the developers at the target software company. This section goes through all the results of all the questions. The individual answers for the open form questions are not explicitly reported.

**Q1 results.** Out of the 27 developers 20 held the job title of a software developer, two were junior software developers, and three software development trainees. One respondent didn't specify their job title and one respondent was a cloud security engineer.

**Q2 and Q3 results.** Out of all the respondents, 37 percent had some previous experience with Dependabot's automated pull requests and 26 percent had experience with some other tool that provides automated pull requests.

**Q4 results.** The practice of dependency vulnerability fixing was relevant for the vast majority of the respondents as is shown in figure 8. Updating dependency vulnerabilities was a recurring task for 78 percent of the respondents with 59 percent needing to carry them out monthly, 11 percent weekly, and 7 percent having to only update yearly.

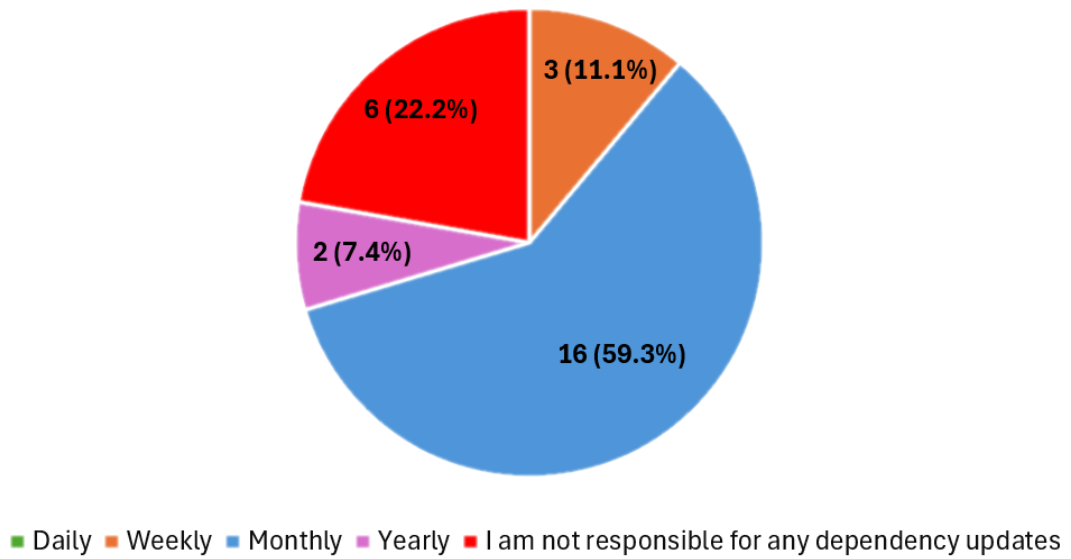


Figure 8. Question 4 results.

**Q5 results.** The increase on awareness of possible security vulnerabilities in project packages was largely agreed upon among the respondents. Over 85 percent of respondents agreed or strongly agreed with automated security pull requests increasing their security awareness. Only two respondents disagreed with argument with one slightly disagreeing and one strongly disagreeing.

**Q6 results.** The automated pull request where also seen to increased the speed at which dependency vulnerabilities can be fixed by the large majority of the respondents. The agreement level was slightly lower compared to the awareness improvement with 89 percent of respondents having some level of agreement. Only one respondent strongly disagreed with automated pull making dependency pull request fixing faster.

**Q7 results.** In par with the previous two questions, the improvement of the overall project security with automated dependency vulnerability updates was agreed in some level by over 90 percent of the respondents. The exact results of Q7 along with the results of Q5 and Q6 are visualized in figure 9

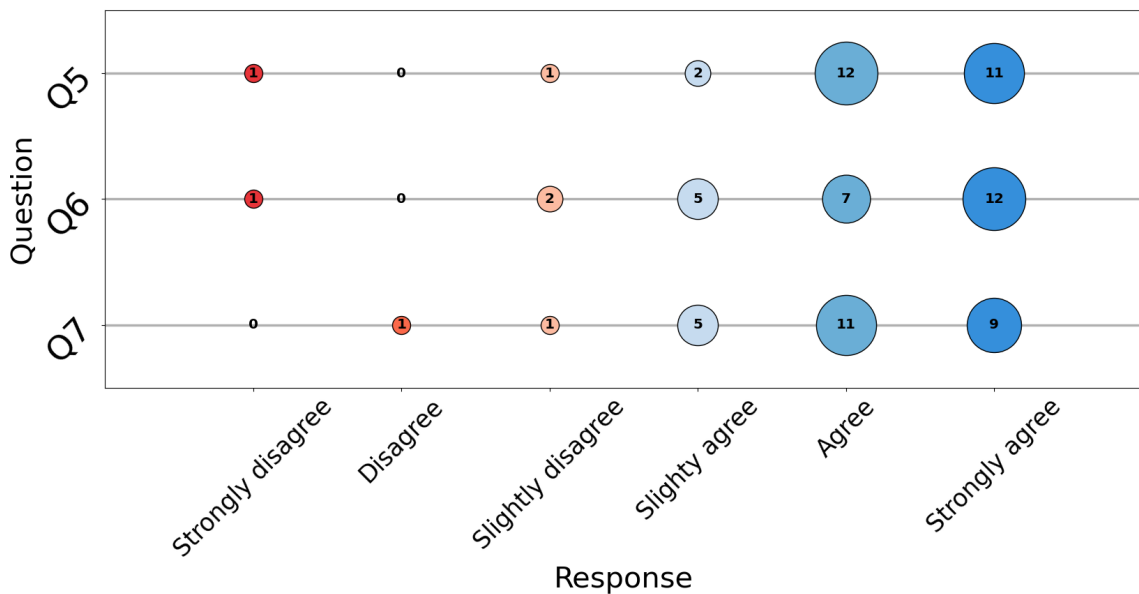


Figure 9. The results of Q5, Q6, and Q7.

**Q8 results.** Out of all the respondents no one was completely unconcerned about the automated pull request introducing breaking changes to the code. On the contrary, 15 percent of the respondents expressed their worries strongly on this manner. However, 51 percent of the respondents were not too concerned with the automated pull request causing additional work by breaking something in their code.

**Q9 results.** The work that needs to be put into reviewing was not seen as effort by the majority of the respondents. 63 percent of the respondents agreed or slightly agreed that a lot of effort is not required to review the automated pull requests. On the other hand 30 percent of the respondents disagreed or slightly disagreed with this statement.

**Q10 results.** The removal of human review and the ability to automatically merge and release the dependency vulnerability fixes saw very little trust. Over 75 percent saw disagreed or strongly disagreed with trusting the pull request to be completely automatically handled even if they had previously not broken anything in the project's code. Only two respondents had slight agreement with trusting this level of automation.

**Q11 results.** Not being familiar with the code was strongly seen as a cause for developers to avoid reviewing the automated pull requests. Over 80 percent of the respondents agreed with this in some level with 26 percent having a strong agreement. Non of the respondents had a strong disagreement with this statement. The exact results of Q1 along with the results of Q8, Q9, and Q10 are visualized in figure 10

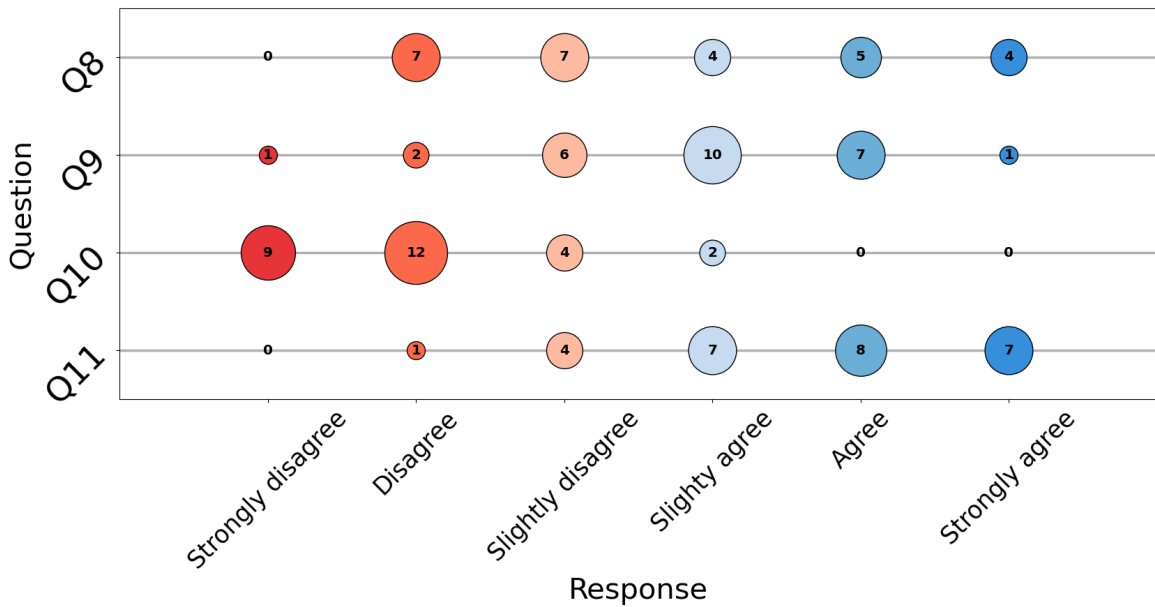


Figure 10. The results of Q8, Q9, Q10, and Q11.

**Q12 results.** Specifically assigning the pull request review to the developer was largely agreed upon to increase the likelihood of reviewing the automated pull request. 37 percent of the respondents strongly agreed with this statement and 51 percent agreed or slightly agreed. Only about 10 percent of the respondents saw no real affect on having the reviews be specifically assigned.

**Q13 results.** Majority of the respondents leaned more to a large amount of dependency vulnerability fixes creating a feeling of overwhelmingness and discouraging the developer from reviewing the pull requests. However about 30 percent of the respondents still disagreed with the statement at some level. Also only two respondents had strong agreement with this.

**Q14 results.** Reflecting the results of the previous question, having to review multiple changes in a single pull request was largely seen more favorable than reviewing the changes in individual pull requests. About 75 percent agreed or strongly agreed with favoring single pull requests. Only one person was strongly against multiple dependencies being fixed in one pull request.

**Q15 results.** Severity of the vulnerability was also seen as an important factor for encouraging developers to not ignore the automated pull request. Almost 50 percent strongly agreed with the statement that severity decreases the likelihood of ignoring the review. Only 15 percent felt in some level that the severity has no effect on the negligence of the pull requests reviews.

**Q16 results.** Reflecting the results of the previous question, the statement that the likelihood that you assess the automated pull request faster if the severity of the vulnerability is higher saw a large support. Over 90 percent of the respondents agreed with this on some level with over 50 percent having a strong agreement. Only two respondents slightly disagreed with the statement. The exact results of Q16 along with the results of Q12, Q13, Q14, and Q15 are visualized in figure 11

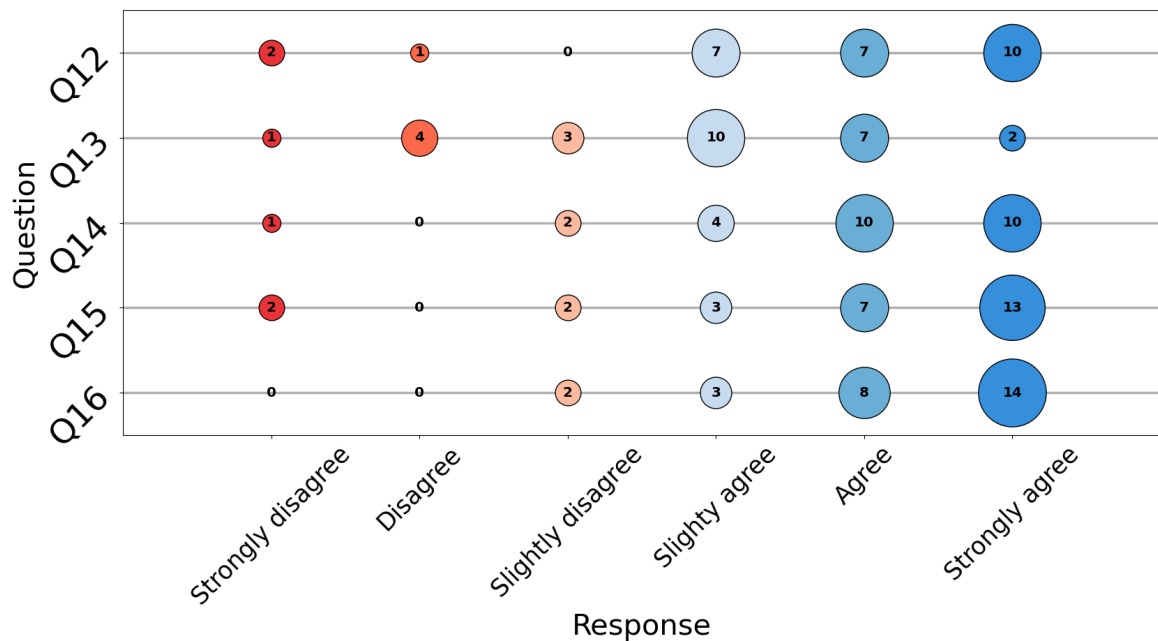


Figure 11. The results of Q12, Q13, Q14, Q15, and Q16.

**Q17 results.** Out of 26 responses the majority estimated the pull request review and release process to not take more than 24 hours to complete. Three respondents estimated the time to take between 24 and 72 hours. Only one respondent estimated it to last over 72 hours.

**Q18 results.** A total of 19 respondents suggested a company specific message application such as Slack as the best way to get notified when a new automated pull request is created by Dependabot. Five respondents also suggested emails as a good way to get notified.

**Q19 results.** As for specific circumstances or issues that would drive developers to avoid reviewing the pull requests, one recurring theme among the answers was the unfamiliarity with the source code the pull request is updating. Being busy was also often mentioned by developers which suggests that other tasks are more likely to have higher priority compared to the automated pull request reviews.

## 6 Discussion

This chapter reflects on findings in the literature review as well as the results of the case study. We evaluate how the conducted research answers the research question and how the findings in this study are related to previous research on the subject. Furthermore, this chapter provides analysis on the results of the case study and discusses the implications of both the literature review findings as well as the case study results. We also propose future research possibilities on the subject.

### 6.1 Implications of the literature review

To answer the **RQ1** this research pointed out three main themes for identifying challenges that may arise when adopting shift-left security into the DevOps lifecycle. When trying to implement shift-left security, it is good to reflect on how the implementations affect the existing security and development practices, how the people in the company and their collaborations are affected, and how the existing tools can be taken into account. On top of accounting for the existing tools and practices, it is important to evaluate how the new shift-left focused tools and practices behave in the company's development environment, and what effects do they have on the speed of the continuous integration and delivery. When the challenges and their themes can be identified, its is then possible to apply solutions to those challenges.

The three themes that where identified for the shift-left security related challenges were also used as the baseline for answering the **RQ2**. The solutions related to practices focused on how traditional and manual security practices such as STRIDE threat modelling and manual package vulnerability fixing could be adapted to allow for them to be utilized in a DevOps pipeline. The main aspect that could be used to enable this was finding ways to automate these practices and to reduce the manual work as much as possible. In addition to implementing the shift-left security practices their performance should be measured and evaluated in order to determine that they do not negatively affect the development efficiency.

Robust collaboration between the development, operation, and security teams was identified as a core component for successful DevSecOps implementation. In order to achieve it, a

continuous security framework can be utilized to ensure that the teams can break out of the isolation and form efficient communication pipelines to provide short feedback cycles on the shift-left security implementations and their effects. To further smooth out the communication between development and security teams, security champions can be utilized.

Despite security tools such as SAST, SCA, and DAST tools being suitable to be implemented into the DevOps development process, especially with the help of IAST solutions, a lot of thought needs to be put on how they are implemented. Poor configuration of these tools can lead to added friction in the efficiency of the CI/CD pipeline. Integrating the use of these tools as early as in the developers IDEs alongside with the automated usage in the pipeline ensures that the security vulnerabilities are detected and fixed as early as possible. This also means that as few vulnerabilities as possible are committed forward to the automated tests and scans in the CI/CD pipeline, thus reducing the time and amount of runs required to ensure secure software.

## **6.2 Implications of the case study**

Since the case study was only conducted inside a single software company with 27 software developers acting as the sample size, the results of the study can not be used to reflect the view on Dependabot in the context of global software engineering field. The case study also only reflects on initial perceptions of the success of the Dependabot implementation, so the results might look different if the research would be done again after an period of active use of the implementation. However, the results can be used to further solidify the findings in previous researches done on the subject. Despite the case study not being able to guarantee the success of the provided Dependabot security update implementation, it can still be used as an entry point for adopting the implementation and further configuring and improving it to fit the specific needs of practitioners.

The first main section of the case study questionnaire strongly supports the claim that the use of automated security pull requests in the manner described in this work do improve the efficiency of updating vulnerable packages. On top of increasing the efficiency, the results also suggest that the overall awareness to vulnerabilities increases with the Dependabot im-



plementation. Both of these results along with the results of Q7 indicate that the Dependabot implementation could have a positive impact on the overall security of the projects implementing it.

The results of the second main section of the case study questionnaire indicated that there is suspicion among developers towards the robustness of the changes introduced by an automated security pull request. The perception on this matter was very evenly divided, which indicates that the trust issues are not an entirely limiting factor for the adaption of the Dependabot security updates. However, the trust issues should still be taken into account when planning on how the Dependabot is implemented to increase the level of trust. This is backed up by the fact that none of the subjects in the study were completely worry free of the trust issues. The suggestion in Alfadel's et al (2021) article to enable the Dependabot to automatically merge security pull requests without a human review saw much higher level of distrust in this study. The results of the Q9 along with the review time estimates of Q17 suggest that the review process of the automated security pull request doesn't require a lot of effort. Based on these findings we argue that it is not recommended to have Dependabot use the auto merge feature. To further streamline the review process and to increase the developers trust towards the pull requests, based on the results of Q11, we suggest assigning developers who are familiar with the specific projects as the reviewers for the pull requests. This suggestion was also backed up by the Q12 results where the vast majority felt that being specifically assigned would increase the likelihood of them not ignoring the pull requests. An example of how this type of reviewer assignments can be configured for Dependabot security pull requests is presented in the appendix A.2. The results of Q18 highly promote the ability to get notified about the pull requests through a company specific communication platform. Having reviewers automatically assigned also enables this functionality to be implemented.

Another identified barrier for automated security pull requests implementation, which also is connected to the trust issues, is the negligence towards the automated pull requests. One major reason for this negligence that was identified in the literature was the notification fatigue. The results for the Q13 also reflected this as the majority felt that large amount of new vulnerability fixes would feel overwhelming and discourage from assessing them. A presented solutions in the literature for this problem was to enable the grouping option for

the automated vulnerability fixes. This would mean that multiple vulnerability fixes could be assessed in a single pull request instead of multiple ones. Based on the results of Q14 this way of grouping was seen as a much more favorable way of dealing with the vulnerability fixes by the respondents. Alfadel et al. 2021 found the severity of the vulnerability to have no significant influence on the how long it takes for the automated pull requests to be handled. Agaronian's (2021) study however noticed that vulnerabilities with higher severity were on average more proactively resolved than those with lower severity. The results of the Q15 and Q16 also supported Agaronian's findings. The results suggested that a higher severity increases the likelihood that an automated pull requests gets reviewed as well as how fast it is done. As recognized in the literature review, the identification of the severity is important for deciding which fixes need to be prioritized and which are trivial. In this study we were unable to find a way to integrate the severity of the vulnerability on the pull request itself, but they are however by default visible in the Dependabot alerts as showcased in appendix A.4. Based on these results the Dependabot security pull requests should not get ignored to the extent that the implementation should be avoided. Using the grouping functionality and integrating the severity into the review process should further decrease the likelihood of the automated pull request getting ignored as well as speed up the reviewing process.

Overall the results and the reviewed literature support the claim that Dependabot's automated security updates can be used to improve the development efficiency in regards of package vulnerability fixing. The provided guideline on how to implement the Dependabot is also applicable for practitioners. The results of the study also suggest that the barriers to the adaption shouldn't end up preventing the use of the implementation. This means that this studied implementation further answers the **RQ2**.

### **6.3 Relevance to previous research on the subject**

In their article "*Challenges and solutions when adopting DevSecOps: A systematic review*" Rajapakse, Zahedi, Babar, and Shen (2022) conducted a systematic literature review where their objective was to identify the challenges that practitioners have faced when adopting DevSecOps. The articles purpose was also to write down and categorize solutions to these challenges that were proposed in the reviewed materials. They reviewed 54 studies and were

able to identify 21 challenges and 31 specific solutions. In the literature review, Rajapakse, Zahedi, Babar, and Shen categorized the identified challenges into four themes that were challenges related to tools, practices, infrastructure, and people. Challenges related to tools were found to be the main research area for DevSecOps. One of the key findings related to that was how traditional tools have drawbacks that prevent them from being optimally utilized in DevSecOps. One of the most practice related challenges that appeared frequently in their research materials was the inability to rapidly and continuously carry out security requirement assessments. Shift-left and continuous security practices were found to be key recommendations for answering these challenges. A critical issue that was pointed out in their article was that many traditional security practices that require manual effort to be conducted are hard to be automated into the development pipeline. The people related challenges and solutions were found to be less discussed in their reviewed materials. Because of that, Rajapakse, Zahedi, Babar, and Shen suggest that further research is needed on people-centric challenges and how they impact the other themes surrounding DevSecOps. Another suggestion for future research in their article was to achieve agreement on initiatives such as implementing shift-left security and determining methods for automating conventional manual security procedures to align with the DevSecOps framework, which is what this works research also continued upon on. While this work's focus with the DevSecOps adaptation is on maintaining the development efficiency of DevOps, the work from Rajapakse et al focuses on the challenges and solutions on a comprehensive level.

Alfadel, Costa, Shihab, and Mkhallalati (2021) investigated numerous GitHub projects with Dependabot enabled. The goal of their study was to find out what are the reasons behind Dependabot security pull requests not getting merged in those projects. Their study also aimed to determine how quickly and how frequently the Dependabot security pull requests are merged as well as what factors have influence over the speed the pull requests get merged. Their results found out that a little over 65 percent of pull requests ended up to be merged out of 13,003 inspected Dependabot security pull requests. Their results also indicated that out of these pull requests the vast majority ended up merged within a day, which is in line with the results of this work. As for the reasons behind merge times they found out that previous experience with Dependabot was associated with fast merge times as well as the use of auto-merging feature. They also found out that in projects with large number of dependencies

it tends to take longer for the pull requests to get merged. Contrary to the results in this work, their research found no severity of the vulnerability to have no significant affect on the merge times. As for reasons for the pull requests to not get merged, excluding cases where Dependabot closes the pull request because it has created a newer pull request that overrides these changes, the most common reason was that the vulnerabilities were manually fixed in another pull request. Their study only analyzed JavaScript projects, which differs from this work in which we didn't account for the language of the projects when conducting the case study.

## **6.4 Future research**

The literature review in this work focused on existing studies on the implementation of DevSecOps and shift-left security in order to point out challenges and solutions related specifically to the development efficiency. These studies focused on the implementations from a more comprehensive viewpoint which is why development efficiency aspect could only be studied to an extent. In order to get more real world data on how the DevSecOps and shift-left security implementations really integrate in practice, we suggest conducting more studies on companies that have implemented these features and evaluating how they have succeeded. This type of study could reveal a larger amount of challenges and solutions for the implementation of DevSecOps from the efficiency viewpoint.

In the case study we discovered that the security update functionality of the Dependabot was lacking in terms of configurability compared to the regular dependency updating functionality. We suggest conducting further research on how the configurability limitations could be bypassed with the Dependabot security updates as well as what the best practices for the configurations might be. Another approach for this research could be to compare other dependency management bots and their security vulnerability functionalities with how they fare in comparison to the Dependabot. The research could also expanded upon to further study the auto-merging features of these bots and if they can in fact be utilized to further increase the advantages of these bots.

This work focused a lot on the security vulnerability automation as a solutions for maintain-

ing the development efficiency in a DevSecOps environment as it was the most beneficial from the view of the case study target company. More detailed research and similar case studies could also be done in the future on the other solutions identified in the literature review chapter of this work. This type of research could also help with further increasing the knowledge on challenges related to those solutions.

## 7 Summary

This work reviewed academic and grey literature to identify various challenges and solutions related to maintaining the development efficiency when implementing shift-left security into the DevOps development pipeline. We identified three main themes for the challenges facing the development efficiency with the shift-left security implementations. These themes were challenges related to practices, people and collaboration, and tools and technologies. We also identified various solutions that can be applied to overcome these challenges.

To further study one of these solutions to see how applicable they are on a real world setting, this work also conducted a case study on the use of GitHub's Dependabot for automated security vulnerability updates. In the case study we created a guideline for how the Dependabot's security updates can be enabled, configured, and assessed. The guideline is showcased in appendix A. The case study was conducted in the setting of a Finnish software development company. The study also created and used a questionnaire to collect insight and prejudices from the target company's developers for the usefulness of the implementation as well as its barriers for adaptation.

The case study yielded 27 responses from the developers of the target company. The results suggested that the usage of automated security pull request would increase the efficiency at which developers can fix package vulnerabilities. Additionally the results also indicated an increase in awareness to the vulnerabilities and in the overall security in the projects. The identified barriers to the automated security pull request adaptation, trust issues toward the pull request changes and negligence to assess them, were also not found to be severe enough to discourage the use of automated vulnerability fixes. Various ways to improve the levels of trust and to reduce the likelihood of negligence were also identified in the study. Based on all these findings the adaptation of the Dependabot implementation as shown in appendix A can be promoted to practitioners.

## References

“5 benefits of shift left security”. 2021. Accessed: 08 February 2024, <https://snyk.io/learn/shift-left-security/>.

Adams, Richard J, Palie Smart, and Anne Sigismund Huff. 2017. “Shades of grey: guidelines for working with the grey literature in systematic reviews for management and organizational studies”. *International Journal of Management Reviews* 19 (4): 432–454.

Adenowo, Adetokunbo AA, and Basirat A Adenowo. 2013. “Software engineering methodologies: a review of the waterfall model and object-oriented approach”. *International Journal of Scientific & Engineering Research* 4 (7): 427–434.

Agaronian, Andrei. 2021. “On Resolution of Vulnerable Dependencies with Dependabot Security Updates in JavaScript Projects”.

Alfadel, Mahmoud, Diego Elias Costa, Emad Shihab, and Mouafak Mkhallalati. 2021. “On the use of dependabot security pull requests”. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 254–265. IEEE.

Anjaria, Dhaval, and Mugdha Kulkarni. 2022. “Effective DevSecOps Implementation: A Systematic Literature Review”. *Cardiometry*, number 24, 410–417.

Bashvitz, Gadi. 2023. “What Is Dynamic Application Security Testing (DAST)?” Edited by Brightsec.com. Accessed: 15 April 2024, <https://brightsec.com/blog/dast-dynamic-application-security-testing/>.

*JFrog Curation Redefines “Shift Left” Security for Enterprise Software Supply Chains*. 2023. <https://www.proquest.com/wire-feeds/jfrog-curation-redefines-shift-left-security/docview/2835830252/se-2>.

Carter, Kim. 2017. “Francois raynaud on DevSecOps”. *IEEE Software* 34 (5): 93–96.

Cois, Constantine Aaron, Joseph Yankel, and Anne Connell. 2014. “Modern DevOps: Optimizing software development through effective system interactions”. In *2014 IEEE international professional communication conference (IPCC)*, 1–7. IEEE.

- Colliander, Camilla. 2022. “Challenges of DevSecOps”. *English. Master’s thesis. University of Helsinki, Faculty of Science*. url: <http://hdl.handle.net/10138/342887>.
- Conklin, Larry. 2024. “Threat Modeling Process”. Accessed: 14 April 2024, [https://owasp.org/www-community/Threat\\_Modeling\\_Process](https://owasp.org/www-community/Threat_Modeling_Process).
- Cronin, Patricia, Frances Ryan, and Michael Coughlan. 2008. “Undertaking a literature review: a step-by-step approach”. *British journal of nursing* 17 (1): 38–43.
- Decan, Alexandre, Tom Mens, and Eleni Constantinou. 2018. “On the impact of security vulnerabilities in the npm package dependency network”. In *Proceedings of the 15th international conference on mining software repositories*, 181–191.
- “DevSecOps and Shift Left Security: A Guide”. No date. Accessed: 07 February 2024, <https://www.software.com/devops-guides/shift-left-devsecops-guide>.
- Ebert, Christof, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. 2016. “DevOps”. *IEEE software* 33 (3): 94–100.
- “Everything You Need to Know About the Likert Scale in 2024”. 2024. Accessed: 28 April 2024, <https://www.surveysensum.com/blog/everything-you-need-to-know-about-the-likert-scale>.
- Garousi, Vahid, Michael Felderer, and Mika V Mäntylä. 2019. “Guidelines for including grey literature and conducting multivocal literature reviews in software engineering”. *Information and software technology* 106:101–121.
- GitHub. 2024a. *Configuring Dependabot security updates*. Accessed: 4 June 2024. GitHub.
- . 2024b. *dependabot/dependabot-core*. <https://github.com/dependabot/dependabot-core>. Version v0.259.0. Accessed: 4 June 2024.
- Gonzalez, Danielle, Paola Peralta Perez, and Mehdi Mirakhorli. 2021. “Barriers to shift-left security: The unique pain points of writing automated tests involving security controls”. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–12.



“Guide to software composition analysis (SCA)”. 2022. Accessed: 12 April 2024, <https://snyk.io/series/open-source-security/software-composition-analysis-sca/>.

Gunja, Saif. 2023. “What is DevOps? Unpacking the purpose and importance of an IT cultural revolution”. Accessed: 29 March 2024, <https://www.dynatrace.com/news/blog/what-is-devops/>.

He, Runzhi, Hao He, Yuxia Zhang, and Minghui Zhou. 2023. “Automating dependency updates in practice: An exploratory study on github dependabot”. *IEEE Transactions on Software Engineering*.

Hilton, Michael, Nicholas Nelson, Timothy Tunnell, Darko Marinov, and Danny Dig. 2017. “Trade-offs in continuous integration: assurance, security, and flexibility”. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 197–207.

“Interactive Application Security Testing”. 2022. Accessed: 25 May 2024, <https://owasp.org/www-project-devsecops-guideline/latest/02c-Interactive-Application-Security-Testing>.

Koskinen, Anna. 2019. “DevSecOps: building security into the core of DevOps”. Master’s thesis.

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

Kuruvilla, Jobin. 2021. “8 big DevSecOps challenges and how to overcome them”. Accessed: 02 February 2024, <https://www.adaptavist.com/blog/8-common-devsecops-challenges-and-how-to-overcome-them>.

Marsal, Jack. 2023. “What is DevSecOps? And what you need to do it well”. Accessed: 29 March 2024, <https://www.dynatrace.com/news/blog/what-is-devsecops/>.

Mohan, Vaishnavi, and Lotfi Ben Othmane. 2016. “Secdevops: Is it a marketing buzzword?-mapping research on security in devops”. In *2016 11th international conference on availability, reliability and security (ARES)*, 542–547. IEEE.

- Morales, Jose Andre, Thomas P Scanlon, Aaron Volkmann, Joseph Yankel, and Hasan Yasar. 2020. "Security impacts of sub-optimal devsecops implementations in a highly regulated environment". In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 1–8.
- Myrbakken, Håvard, and Ricardo Colomo-Palacios. 2017. "DevSecOps: a multivocal literature review". In *Software Process Improvement and Capability Determination: 17th International Conference, SPICE 2017, Palma de Mallorca, Spain, October 4–5, 2017, Proceedings*, 17–29. Springer.
- Nawale, Mahesh. 2022. "The Top Challenges Faced by Organizations Implementing DevSecOps". Accessed: 02 February 2024, <https://www.zscaler.com/blogs/product-insights/top-challenges-faced-organizations-implementing-devsecops>.
- Nemoto, Tomoko, and David Beglar. 2014. "Likert-scale questionnaires". In *JALT 2013 conference proceedings*, 1–8.
- Nugraha, Muhammed. 2024. "Integrating STRIDE Threat Model With DevOps". Accessed: 02 April 2024, <https://www.practical-devsecops.com/integrating-stride-threat-model-with-devops/>.
- Patni, Aditya. 2024. "Core DevSecOps Challenges & Best Solutions for 2024". Accessed: 02 February 2024, <https://www.practical-devsecops.com/devsecops-challenges/>.
- Planning, Strategic. 2002. "The economic impacts of inadequate infrastructure for software testing". *National Institute of Standards and Technology* 1.
- Rajapakse, Roshan N, Mansooreh Zahedi, M Ali Babar, and Haifeng Shen. 2022. "Challenges and solutions when adopting DevSecOps: A systematic review". *Information and software technology* 141:106700.
- "Security Champions". 2022. Accessed: 02 April 2024, [https://owasp.org/www-project-security-culture/v10/4-Security\\_Champions/](https://owasp.org/www-project-security-culture/v10/4-Security_Champions/).

Sharma, Piyush. 2021. “The future of shift-left security is infrastructure-as-code: Piyush Sharma, CEO & Co-Founder, Accurics by Tenable”. *Express Computer*, <https://www.proquest.com/trade-journals/future-shift-left-security-is-infrastructure-as/docview/2615125143/se-2>.

Shull, Forrest, Janice Singer, and Dag IK Sjøberg. 2007. *Guide to advanced empirical software engineering*. Springer.

Soni, Mitesh. 2016. *DevOps for Web Development*. Packt Publishing Ltd.

*State of DevSecOps in Europe*. 2023. Technical report. synopsys. <https://www.synopsys.com/software-integrity/resources/analyst-reports/state-of-devsecops.html>.

“Static Application Security Testing”. No date. Accessed: 28 March 2024, <https://www.synopsys.com/software-integrity/resources/analyst-reports/state-of-devsecops.html>.

Wessel, Mairieli, Tom Mens, Alexandre Decan, and Pooya Rostami Mazrae. 2023. “The GitHub development workflow automation ecosystems”. In *Software Ecosystems: Tooling and Analytics*, 183–214. Springer.

“What is Software Composition Analysis?” No date. Accessed: 28 March 2024, <https://www.reverera.com/blog/what-is-software-composition-analysis/>.

Wu, Xiaojun, Anze Gao, Yang Zhang, Tao Wang, and Yi Tang. 2022. “A preliminary study of bots usage in open source community”. In *Proceedings of the 13th Asia-Pacific Symposium on Internetware*, 175–180.

www.castsoftware.com, editor. No date. “What is Software Composition Analysis (SCA)?” Accessed: 12 April 2024, <https://www.castsoftware.com/glossary/what-is-software-composition-analysis>.

Zhou, Xin, Runfeng Mao, He Zhang, Qiming Dai, Huang Huang, Haifeng Shen, Jingyue Li, and Guoping Rong. 2023. “Revisit security in the era of DevOps: An evidence-based inquiry into DevSecOps industry”. *Iet Software* 17 (4): 435–454.

## Appendices

### A Guideline for using Dependabot to automate dependency vulnerability fixing

Dependabot is a GitHub native dependency management bot that can be used to keep dependencies up to date and to detect vulnerable dependencies and automatically create fixes for them. Dependabot security pull requests update the dependency to the first version where the vulnerability is fixed, NOT to the latest version.

#### A.1 Enabling Dependabot

Enabling Dependabot automated security updates can be done easily in the GitHub repository's settings, by enabling **Dependency graph**, **Dependabot alerts**, and **Dependabot security updates**. To allow Dependabot to group all automated security updates into a single pull request, the **Grouped security updates** can be enabled. The settings are shown below

Wiki Security Insights **Settings**

**General**

Access

- Collaborators and teams
- Team and member roles

Code and automation

- Branches
- Tags
- Rules
- Actions
- Webhooks
- Copilot
- Environments
- Codespaces
- Pages
- Custom properties

Security

- Code security and analysis**
- Deploy keys
- Secrets and variables

Integrations

- GitHub Apps
- Email notifications
- Autolink references


## Code security and analysis

Security and analysis features help keep your repository secure and updated. By enabling these features, you're granting us permission to perform read-only analysis on your repository.

**Dependency graph** Disable  
Understand your dependencies.

**Dependabot**  
Keep your dependencies secure and up-to-date. [Learn more about Dependabot.](#)

**Dependabot alerts** Disable  
Receive alerts for vulnerabilities that affect your dependencies and manually generate Dependabot pull requests to resolve these vulnerabilities. [Configure alert notifications.](#)

**Dependabot rules** 0 rules enabled   
Review and manage alert presets.

**Dependabot security updates** Disable  
Enabling this option will result in Dependabot automatically attempting to open pull requests to resolve every open Dependabot alert with an available patch. If you would like more specific configuration options, leave this disabled and use [Dependabot rules](#).

**Grouped security updates** Disable  
Groups all available updates that resolve a Dependabot alert into one pull request (per package manager and directory of requirement manifests). This option may be overridden by group rules specified in dependabot.yml - [learn more here](#)

**Dependabot version updates** Enable  
Allow Dependabot to open pull requests automatically to keep your dependencies up-to-date when new versions are available. [Learn more about configuring a dependabot.yml file.](#)

**Dependabot on Actions runners** Enable  
Run Dependabot security and version updates on Actions runners.

**Dependabot on self-hosted runners** Enable  
Run Dependabot security and version updates on self-hosted Actions runners.

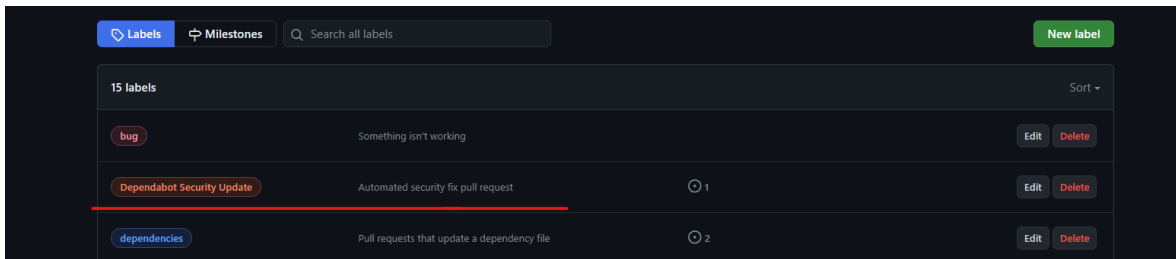
## A.2 Configurations for the security pull requests

Dependabot allows a lot of configuration options for the dependency updates, but unfortunately these don't really work with the security updates. However, GitHub actions can be used to add additional configuration for the automated security updates. A pre-made GitHub action can be found in the section Dependabot configuration and usage. The action detects the Dependabot created security update pull requests, and adds a custom label and assigns reviewers to be always assigned to it. Assigning reviewers to the automated pull requests ensures that the teams/developers responsible for the repositories are immediately notified of a new vulnerability fix.

**NOTE:** I was unable to find solutions to issues regarding the use of GitHub teams as reviewers through the **team\_reviewers** option which would have been a more desired configuration. So for now the solution is to just individually assign the reviewers and update them. In the future, the desired solutions would be to be able to assign Teams as reviewers.

**NOTE:** Also be careful to not confuse this with the actual Dependabot.yml file, so **avoid** naming the custom configuration file **Dependabot.yml**

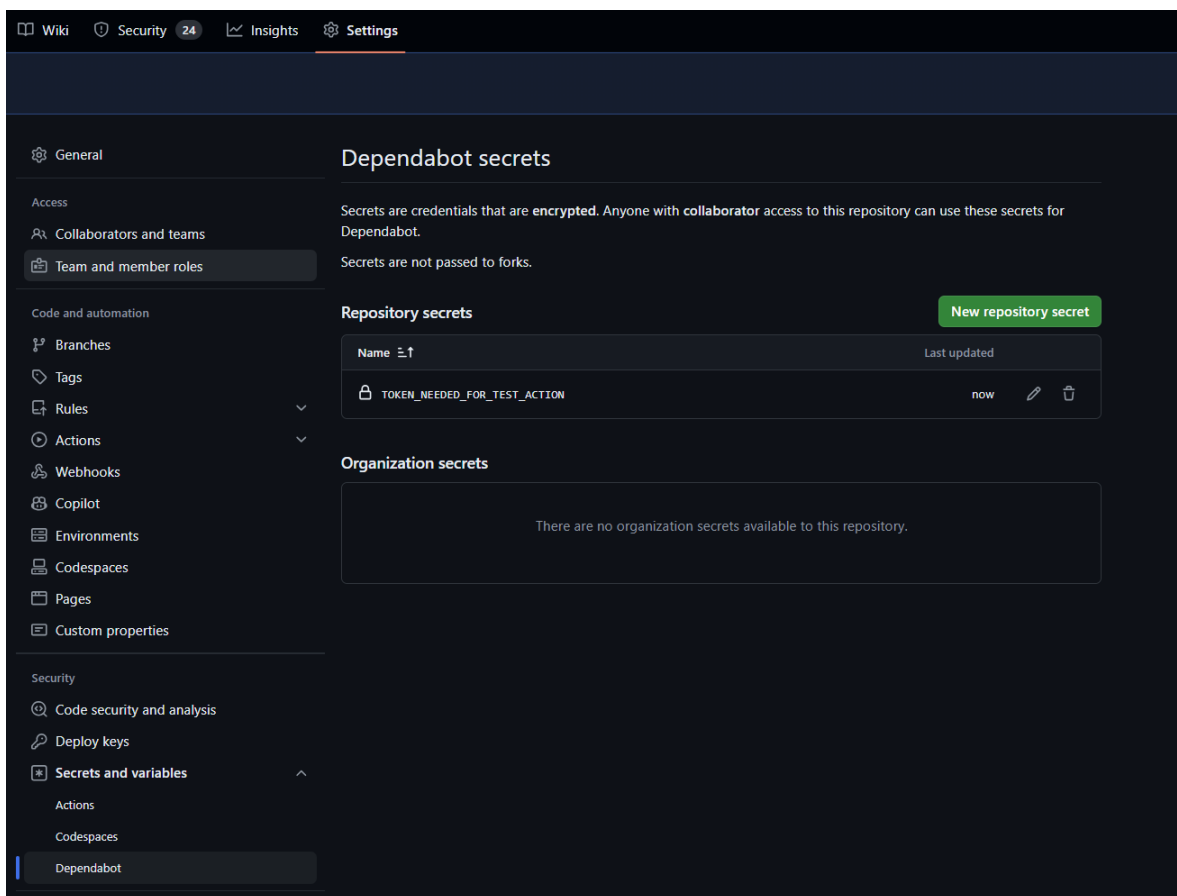
The security label specified in the GitHub Action is automatically created if it does not exist in the repository's labels. To change the colour of the label to be more alerting and to add custom info, the label can be added/edited in the repository's labels. Here is an example of the label used in the template.



Dependabot also automatically creates a **dependencies** label that it adds to all automated security updates PR's. do not alter this label, as it's used to identify the automated security PR's.

### A.3 Access to repository secrets needed for actions initiated by Dependabot created pull requests

By default, the Dependabot created pull requests do not have access to the **Secrets and variables** that are present for GitHub actions in the repository's settings (**Actions** under **Secrets and variables**). If you have actions that are run on every created pull request (i. E. tests) that require those secrets, the secrets need to also be added to the **Dependabot** section in the **Secrets and variables** in the repository's settings.

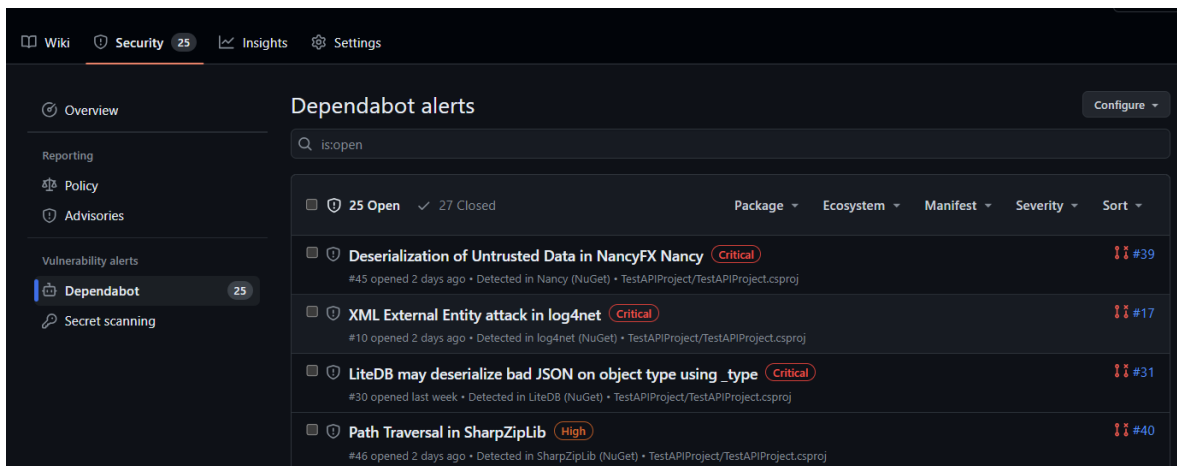


## A.4 Assessing the automated security pull requests

The Dependabot will attempt to create a new security pull request every time a new vulnerability is found on some of the packages used in the project or every time a vulnerable package is pushed to the main branch of the project. If the **Grouped security updates** option is turned on, all unfixed dependency vulnerability fixes are automatically merged into a single pull request, making it easier and less time consuming to verify and release the changes.

You can navigate to **Security -> vulnerability alerts -> Dependabot** in the GitHub repository to see all the unfixed vulnerabilities in the project as well as the severity of the vulnerability. This page can be used to quickly determine how urgent it is to review and release the dependency updates. (example shown below)

**NOTE:** The **Dependabot alerts** page shows if the vulnerability is linked to a fix in a pull request, but these might not properly update when different PR's are grouped together by Dependabot and show closed PR's. So don't blindly trust them.



The Dependabot pull request shows details on what package is to be updated and to what version. It also shows release notes (if available) on all the versions between the current release and the one that is to be updated. These release notes can be quickly investigated to see if they introduce any breaking changes.



When you have reviewed the dependency update pull request enough to determine that you think it won't break anything, it is recommended to deploy the branch and see that everything builds correctly.