TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Dachi Mshvidobadze 201818IVSB

Case Study of Automated Vulnerability Management of Microservices at Pipedrive OÜ

Bachelor Thesis

Supervisor:	Kaido Kikkas
	PhD
Co-Supervisor:	Renno Reinurm
	Bsc

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Dachi Mshvidobadze 201818IVSB

Pipedrive OÜ Mikroteenuste Turvanõrkuste Automaathalduse Juhtumiuuring

Bakalaureusetöö

Juhendaja:	Kaido Kikkas
	PhD
Kaasjuhendaja:	Renno Reinurm
	Bsc

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Dachi Mshvidobadze

04.17.2022

Abstract

This thesis aims to analyze how the additional security checks recently implemented against software source code updates at Pipedrive OÜ affect the efficacy of software engineers. This is done by comparing time and effort needed to implement changes in the code before and after the security checks have been enforced. Alongside this, the thesis will look into ways of improving the effectiveness of tools used for 3rd party code package management inside Pipedrive's code repositories, with the aim of lowering the amount of human supervision needed.

Deployment pipeline is a set of automated procedures that run against every new revision of software, before the revision can make it into the live product. The procedures can include automated compilation checks, automated software tests, static checks for code style, etc. Such a pipeline has been present at Pipedrive for a long time, and the newest addition to it, aimed at automating the management of vulnerabilities, is a set of security checks that run against the software source code and the software packages that the code uses. This creates additional requirements that the software engineers must fulfill before their code can be pushed into production.
Additionally to the deployment pipeline, there are other tools that can automate vulnerability management. Of interest to this thesis are a type of tools that help keep 3rd party code free of vulnerabilities, by automatically suggesting which version of the code to use. However it is not always clear how smoothly the different versions comply with the existing codebase, thus requiring human supervision.

The thesis is in English and contains 26 pages of text, 5 chapters, 7 figures.

List of abbreviations and terms

API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Delivery
CPU	Central Processing Unit
CRM	Customer Relations Management
Deployment Pipeline	a deployment pipeline is a system of automated processes designed to quickly and accurately move new code additions and updates from version control to production
E2E	End To End
Git	A free and open source distributed version control system
GPL	Gnu Public License
IDE	Integrated Development Environment
ISO	International Organization for Standardization
Microservice	Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs
PR	Pull Request
SAST	Static Application Security Testing
SemVer	Semantic Versioning
SOC	Security Operations Center
SWE	**S**oftware **E**ngineer/ing
VM	Virtual Machine

Vulnerability	A weakness of an asset or group of assets that can be exploited by one or more threats, where an asset is anything that has value to the organization, its business operations, and their continuity, including information resources that support the organization’s mission

Table of Contents

1 Introduction

Pipedrive is an Estonian late stage startup, founded in 2010[1]. Its business model can be characterized as Software as a Service, and the software it sells is a Customer Relations Management (CRM) application, with a subscription-based business model. As recently as in 2021 it was acquired by an investment firm[2] and secured 1 billion USD, thus becoming a "unicorn". This marked the gradual transition out of being a startup, into a more stable structure. This came with slow but noticeable changes, such as standardization and stabilization of processes, many of them being security processes.

One of those security process changes has touched the deployment pipeline[3] of microservices[4]. This pipeline is set up on the code repositories of all active microservices, it so far involved only automated testing of software and checks on code styles and guidelines. The change in question is the addition of security checks into the pipeline - which means that the code now has to pass software tests that are written by the developers, the code quality checks that are configured also by the developers, and also security checks need to be satisfied - this is set up with the supervision of security team at Pipedrive. This is one of the many automated vulnerability management processes in use at Pipedrive.

1.1 Structure of the Thesis

Chapter 1 describes the problem, the scope and the goals of the thesis. Chapter 2 provides the background information about what vulnerability management, in general is, what is it like in Pipedrive, and the technologies used for microservice vulnerability management in Pipedrive. It also discusses the state of the art and how it relates to the process in Pipedrive. Chapter 3 will formulate the methodology of the thesis, and talk about expected results. Chapter 4 will focus on practical analysis, its results and their interpretation. Finally, chapter 5 will conclude and summarize the thesis, as well as provide possibilities of further improvement.

1.2 Description of the problem, Scope of the thesis, and Formulation of possible solution

As it stands right now, Pipedrive is slowly improving the vulnerability management tool belt that it uses for its microservice source code. The question this thesis asks and tries to answer is: how much does the new processes improve the vulnerability management? Does it have an effect on the speed of development? These questions in turn raise other questions: By what metric will the process be more effective? Which internal teams will have what parts of the process assigned to them? Is the process going to be able to discover more vulnerabilities than the process before? What tools will be used? Will fixing a vulnerability break the software? Can we detect such software breaks, and if yes, how efficiently can that be done? Metrics need to be defined by which one can characterize the effectiveness of developers and its possible changes. The vulnerability management solution, and its parts, need to be described to understand why it is actually implemented.

The problem statement can formally be formulated as follows:

- **RQ1**: Did the additional security checks slow down the development?
- **RQ2**: How do automated tests help with detecting broken code, introduced by bots that manage vulnerabilities?

Scope of the thesis

Scope of this thesis includes describing the microservice vulnerability management process in Pipedrive, how the components of the process fit together and why are they chosen the way they are chosen. The thesis will analyze Github code repositories and the pull requests, through which changes are made in the microservices and will focus on feature additions as well as automated dependency version updates. Out of scope are specifics of vulnerabilities, services and any code on which the analysis is performed, other than specifying the language and ecosystem in use. Also out of scope is maintenance of services on individual basis and any sort of compliance enforcement. Thesis will not include raw data that was used for analysis and won’t include details and specifics of any codebase.

2 Background Information

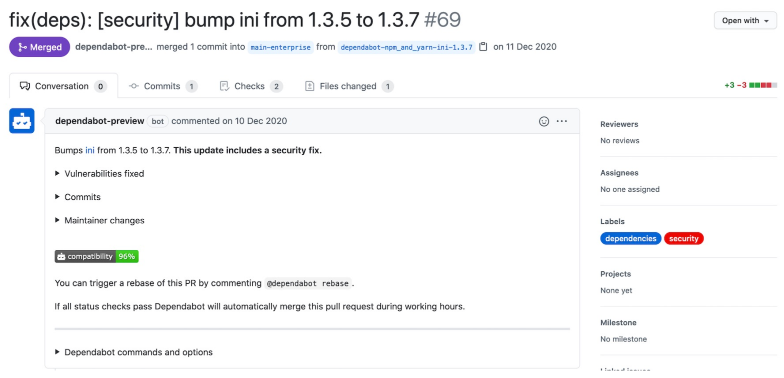
This chapter will give an overview of what are vulnerabilities, what is vulnerability management, why should vulnerabilities be managed within Pipedrive, what are current automation options and what might be the benefits of automating the vulnerability management process.

This chapter will give an overview of what tools, technologies and conventions are used in the vulnerability management process in Pipedrive, then describe what vulnerabilities are defined as, according to the standards against which Pipedrive is certified, and then will give an overview of vulnerability management in Pipedrive, as well as highlight where the mentioned tools fit, and will justify the choices based on scientific literature.

2.1 Tools, technologies, conventions

2.1.1 Dependabot

Dependabot is a freely available vulnerability patching solution that every account on Github can make use of[5]. It uses several open source registries of vulnerabilities for different languages that it covers, including two of the major ones used in Pipedrive - Go and JavaScript, in which it stores which packages have what vulnerabilities and for which versions. This information is used to detect the vulnerable versions of dependency packages in git repositories hosted on Github, and updated. The update is made in a form of a Pull Request. One caveat the pull request based contribution has is that sometimes the pull requests are outright ignored, and Dependabot is largely configured to only create a few branches, 2 by default, in order not to crowd the pull request list of the repository. This creates a problem where a developer either needs to merge two Dependabot updates, or attend to them separately, both of which require time.


fix(deps): [security] bump ini from 1.3.5 to 1.3.7 #69

Merged dependabot-pre... merged 1 commit into `main-enterprise` from `dependabot-npm_and_yarn-ini-1.3.7` on 11 Dec 2020

dependabot-preview bot commented on 10 Dec 2020

Bumps ini from 1.3.5 to 1.3.7. **This update includes a security fix.**

▸ Vulnerabilities fixed

▸ Commits

▸ Maintainer changes

compatibility 96%

You can trigger a rebase of this PR by commenting `@dependabot rebase`.

If all status checks pass Dependabot will automatically merge this pull request during working hours.

▸ Dependabot commands and options

Reviewers: No reviews

Assignees: No one assigned

Labels: dependencies security

Projects: None yet

Milestone: No milestone

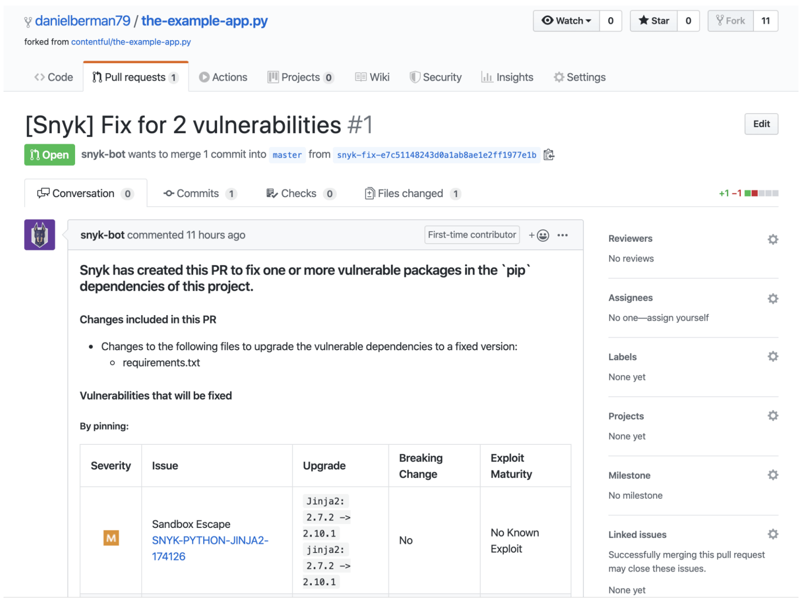
Figure 1. Dependabot security Pull Request

2.1.2 Snyk

Snyk is a toolchain that includes several technologies in it. While some of them are not used in Pipedrive due to other tools of the exactly same nature already being present and in active use, they will still be mentioned for the sake of completeness.

Snyk Open Source Vulnerability Scanning

Much like Dependabot, Snyk open source scanning detects vulnerable dependency packages based on open source registries, and suggests updates. Dependabot has one advantage over it - since Dependabot is owned by Github, it is well integrated into the ecosystem and the updates can be configured easily, and troubleshooting is easy due to the larger community. However, unlike Dependabot, it can be integrated into Pull Requests, hence being able to detect vulnerable packages before they are merged to the main branch and subsequently pushed to production. It also, unlike Dependabot, comes with proprietary metric that helps with vulnerability prioritization, a web user interface that illustrates which git repositories have vulnerable packages and at which level - dependencies have their own subdependencies - and allows one to generate vulnerability reports, that contain vulnerability scores, display which vulnerabilities from which vulnerability registries are going to be fixed with the update and document the amount of possible breaking changes.[6]



[Snyk] Fix for 2 vulnerabilities #1

Open snyk-bot wants to merge 1 commit into `master` from `snyk-fix-e7c51148243d0a1ab8ae1e2ff1977e1b`

snyk-bot commented 11 hours ago

Snyk has created this PR to fix one or more vulnerable packages in the `pip` dependencies of this project.

Changes included in this PR

- Changes to the following files to upgrade the vulnerable dependencies to a fixed version:
  - requirements.txt

Vulnerabilities that will be fixed

By pinning:

Severity	Issue	Upgrade	Breaking Change	Exploit Maturity
M	Sandbox Escape SNYK-PYTHON-JINJA2-174126	`Jinja2: 2.7.2 -> 2.10.1` `jinja2: 2.7.2 -> 2.10.1`	No	No Known Exploit

Reviewers
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Linked issues
Successfully merging this pull request may close these issues.
None yet

Figure 2. Sample Snyk vulnerability report

Snyk Open Source License Scanning

License scanning helps detect dependencies and subdependencies which might have constrictive licenses, such as GPL 3.0, that need attention, since the breach of license can be grounds for a legal action. Hence, the legal aspects can be brought to attention without the need of manual inspection from humans.[7]

Snyk SAST

Static Application Security Testing "is a vulnerability scanning technique that focuses on source code, bytecode, or assembly code."[8] Used sometimes in the IDEs by the developers, or in most cases in the Github Pull Requests. It detects insecure code by statically analysing the code and highlighting it. Of course the detected code might be a false flag, but that is negotiated with Information Security team, and if threat is not found in the code, then the detection can be ignored.[8]

Snyk Container Image Scanning

This tool scans container images, that are meant for Kubernetes cluster, for vulnerabilities that might come from packages used in the container, it might be a powerful tool but Pipedrive is already using a free and open source alternative - Trivy[9]. Container vulnerabilities will be discussed with it.[10]

2.1.3 Github

Github is a git hosting platform that has a number of integrations and tools available. This section is going to cover several of those integrations and tools that are used in vulnerability management at Pipedrive.

Github Actions

Github actions are a git repository automation tool for repositories active on Github. It can create, merge, delete branches, execute scripts, such as the above mentioned Snyk scans, and in general is a Swiss army knife type of tool used for repository automation. Almost every other tool's presence and operation in a git repository is governed by this tool.

Pull Request

Pull requests are the tool through which incremental updates to repositories happen in Pipedrive. Pull requests include the description of feature update, a list of diverging commits from the main git branch, comments from developers and mainly automation run by either Github Actions, Jenkins, or any other tool that does actions on the code.

Github API

A tool which is not a part of Pipedrive's vulnerability management, but a tool used in this thesis to analyze data about repositories. More on its use in the next chapter: Methodology.

2.1.4 Jenkins

Jenkins is a tool which is famous for being the quintessential Swiss army tool of automation, it was used by nearly every engineering team in Pipedrive, however the developer teams are slowly transitioning from Jenkins to Github Actions. Function of Jenkins that is relevant to the thesis, is that it was able to run any kind of test suite, including the ones that required building Kubernetes clusters, such as functional tests.

2.1.5 Tests

Tests are an integral part of microservice development in Pipedrive, as they help raise the quality of the code by making sure that the code works as intended, help find breaks in functionality during the development. Select test suites are run automatically at select points in the continuous delivery process thus making the it more robust, and making sure that there is no broken code in the live environment. There are several types of tests, and the ones present in Pipedrive will be discussed.

Unit Tests

Unit tests cover execution of isolated methods and are the simplest tests that exist, hence they require the least amount of resources to run. Because of this, they are run in pull requests on every update.

Functional Tests

Functional tests are a type of integration tests used to test API calls within back-end microservices. More often than not they require several components that need to be set up, not only do parts of the application have to be mocked, but also mock services need to be deployed, which respond to pre-determined responses to requests going in from the tested microservice. This requires extra containers in which the mock services will live. This means extra resources, due to which these kinds of tests are not automatically run like the unit tests. In Pipedrive's environment, the developers run these tests near the end of a singular feature development round.

Integration Tests

Mentioned as courtesy, Integration tests have their purpose in the name - they test integration of several components together. Functional Tests are one such type of test.

2.1.6 End to End or E2E Tests

These are the most resource-intensive tests available - they test how a customer would interact with the application as a whole. It simulates front-end interaction and expects the application to react in a certain way, and reports the mismatches exactly. This requires the presence of the entire microservice stack in an isolated environment. Due to the intensity of the required resources,

Smoke Tests

Smoke tests are a subset of E2E tests which only test the "happy path interaction" or an interaction of a user with an app which should, in theory, not cause any errors. These tests are not a part of development process, and are ran separately and quite often, to ensure that nothing is broken.

Code Coverage

Code Coverage is a metric which shows how many lines of code does a test suite cover. This is one of the metrics that are going to be used during the practical part - expectation is that higher code coverage levels should result in reporting broken code more often.

2.1.7 SemVer - Semantic Versioning

Not a tool, but a convention for versioning software. It is used in most of the dependency packages used at Pipedrive. Brief synopsis of this convention is that versions take the form X.Y.Z where X would stand for a "Major Version" - which means there are breaking changes, such as removal of deprecated API methods. Y stands for "Minor Version" where the changes made are backwards compatible. Z is "Patch Version" where bugs are fixed in a backwards compatible manner, but no changes introduced.[11]

2.2 Vulnerability Management Concepts

As defined in ISO 27005[1], a vulnerability is "A weakness of an asset or group of assets that can be exploited by one or more threats, where an asset is anything that has value to the organization, its business operations, and their continuity, including information resources that support the organization's mission".[12, 13] Vulnerability Management, in turn, is defined as "the process in which vulnerabilities in IT are identified and the risks of these vulnerabilities are evaluated. This evaluation leads to correcting the vulnerabilities and removing the risk or a formal risk acceptance by the management of an organization."[14] These are of course broad definitions, and they require more precise scoping, this thesis looks at vulnerabilities in microservice source code, the 3rd party code (i.e. - dependencies) used in the microservices, as well as the dependencies of the dependencies.

Aforementioned tools are used with a certain framework, to maximize the effectiveness of the limited resources that the Information Security team has. The automatic Snyk SAST and Open Source Dependency Vulnerability checks make sure that the code pushed to production is as secure as possible, while Dependabot generates updates itself for the repository owners. This step offloads a lot of burden to the developers, since Information security team does not have to normally take part in these activities. This is already a strong improvement over the baseline, as in 2017 a study has found that 81.5% of the studied systems were keeping outdated dependencies because of the perceived extra workload and responsibility. As a part of the same study, 69% of the surveyed developers claimed that they were unaware of their vulnerable dependencies.[15] However, it needs to be said that this study was conducted on repositories with Java code, and while language differences don't usually matters, one of the study's conclusions was that a project was more likely to update if it had a dependency management tool, which most modern JavaScript projects, and all JavaScript projects in Pipedrive, already have in the form of node package manager. Further proof that this delegation of vulnerability management to the developers is a good idea, is that according to a study done by Danny Dig, et al.[16] 84-97% of breaking API updates are refactorings, i.e. change of existing code. This is exactly the category under which security updates to dependencies fall - they change, which means they refactor, existing code, and if the changes in APIs actually breaks the code, then it will fall onto the developers who are the owners of the microservice to fix the issue - hence, their involvement is mostly inevitable. And while earlier described Semantic Versioning can be used to differentiate between breaking and non-breaking API changes, a study done by Bogart Christopher, et al.[17] that updates that do not respect SemVer, do happen - this makes trusting the authors somewhat less risk-free. Another reason why it's a good

[1]Pipedrive is certified against ISO27001 which in term derives the definition from ISO27005, due to this fact, this thesis takes the same definition

idea to let the developers handle vulnerability updates in such a way, was illustrated in the study done by Laerte Xavier, et al.[18] which discovered that the more time passed, the more breaking changes were introduced to an API, which meant that upgrading one's dependencies should be done somewhat often, in order to not accrue technical debt.

For the sake of completeness, the human-driven process will also be documented:
As stated above, the Snyk Open Source tool, along with Trivy, are used to monitor the vulnerabilities in timely fashion - these are one of the tools that help Pipedrive stay compliant with ISO27001, namely the 12.6.1 clause, which states that "Information about technical vulnerabilities of information systems being used shall be obtained in a timely fashion, the organization's exposure to such vulnerabilities evaluated and appropriate measures taken to address the associated risk."[12]. Snyk reports, that include severity and the maturity of the exploits, cut down on the time that is needed for the SOC team to prioritize some vulnerabilities over others. Trivy reporting allows for visualizing and pinpointing the container image vulnerabilities. The vulnerabilities that appear are submitted to the developer teams, that own the microservice, in a form of an agile board ticket with a priority label - the more severe the vulnerability, the higher the priority, the sooner it has to be fixed. Development teams can discuss with information security team the severity of the vulnerability and the priority can be changed, however all the vulnerabilities are addressed, sooner or later.
The earlier mentioned Github Actions are used to automate the delivery of Dependabot pull requests. The unit tests are run even for the Dependabot pull requests, and if the project can be successfully built and the tests successfully pass, the project automatically gets deployed to the pipeline, where the most resource-intensive tests, the End-to-End tests are run, and if they are also successful, then the microservice with the code updated by Dependabot is working in live environment.

Not to be overly dependent on just the automated tools, Pipedrive tries to insulate itself from the threats by having another layer of vulnerability detection - in partnership with HackerOne, Pipedrive runs a bug bounty program, through which security researchers can submit vulnerability reports of exploits found in the Pipedrive's product. It is not only intuitive that leaving the vulnerability detection process to the automated tools, but is also detected by security researchers at TU Crete[19] that Snyk tests and npm audit might miss some vulnerabilities in source code, such as zero-day vulnerabilities.

2.3 Motivation

"The only system which is truly secure is one which is switched off and unplugged, locked in a titanium lined safe, buried in a concrete bunker, and is surrounded by nerve gas and very highly paid armed guards. Even then, I wouldn't stake my life on it..."
Gene Spafford - Director, Computer Operations, Audit, and Security Technology (COAST), Purdue University.[20]

Implementing more tools into the workflow can of course improve security, but it has a toll on the time of feature delivery, in that the workflow gets more complicated: each new feature addition must comply with several kinds of tests, thus it can be assumed that they require additional attention and time from the developers. Assessing monetary side of developer time usage is beyond the scope of this topic, however the question of security versus usability is of interest to all parties involved, so this thesis will look at this workflow update from exactly that lens - how much is being added to the security and how much does it complicate matters? The security versus usability is an ever-discussed topic within the field of security.
Current workflow of software automated security testing and open source vulnerability scanning undoubtedly require attention from the developers, therefore require their time. This means that every single feature update will take more time, and what is more important - it will take time that is not being accounted for while planning, since the feature is so new. There were already checks in place for the developers to comply with information security requirements. Those are in the form of a meeting with information security team, but no automated code checks. This means that the possible time increment in development process can be investigated.

Another aspect of the added tooling - Github Actions automated Dependabot updates - has a small caveat. If several Dependabot updates are triggered at once, the internal deployment tool[21] might get overloaded, since the infrastructure has limits. Also need to be noted, that in 2021 study by Joseph Heidrup, et al.[22] that "that tests can only detect 47% of direct and 35% of indirect artificial faults on average." - This means that potentially, there might be a change that slips into production that causes some sort of instability.

Improving security is often associated with heightened complexity of the process, due to requirements that need to be first understood, then satisfied during the process, which needs to be reviewed and addressed, resulting in more time being lost. A good example would be passwords - the more complex a password is the harder it is to brute-force the access to an account the password is protecting, but at the same time it's harder to remember, so humans tend to write them down, or use some other "lazy solution"[23]. Although it can be speculated that such effects can possibly be observed in Pipedrive's microservice code, there is a code quality assessment procedure in Pipedrive by a peer from the same team, that distributes the responsibility onto the peer as well. This practice has been proven to improve the overall quality.[24]

3 Methodology

To answer **RQ1** and **RQ2** earlier mentioned Github API is going to be used to collect relevant data, paired with SPoT (Single Point of Truth)[21] - it was not mentioned earlier since it is not a part of vulnerability management process, but contains important information about Pipedrive's microservices, of interest to this research is code coverage. Since SPoT contains data on all repositories of Pipedrive, it allows us to take those repositories which do have code coverage in them, due to the presence of repositories (automation tools, infrastructure code, etc.) which by their nature can not have the same kinds of tests as microservice repositories.
Quantitative analysis of the repository data will be performed to assess how different characteristics of a pull request are correlated to each other, such as:

- Total size of a pull request - lines added/removed, number of commits
- Total time open - time of creation, merging, closing
- Who opened the PR - was it a human, or a robot?

Correlation between these metrics can show how much effort was put into a feature update - how much code was written? and over how many commits? It is also expected to infer how quickly were the changes made from the amount of time that the pull request was open. Of interest will be also the pull requests that are left open still, especially if they're opened by a bot - this would mean that a dependabot pull request was either not attended to, or that there was already an update in the dependency and the pull request just timed out - such cases have been recorded[25]. Figure 3 Describes the flow of the data that is used in this part of assessment.
It is expected that since the introduction of additional checks, the average time and effort put into a pull request should increase. The dataset consists of 2477 pull requests and 52 repositories.

To suggest improvements, a qualitative study of select repositories can be done. In previous chapter, Functional tests were described as being manually run - this means that they are not automatically run in Dependabot pull requests, which means that if inconsistencies are not caught at build or unit test stages, which do not cover the full spectrum of the microservice's operation, then a far costlier End-to-End tests, as opposed to functional tests, are going to be run. This has both pros and cons: if the build passes, then a Dependabot pull

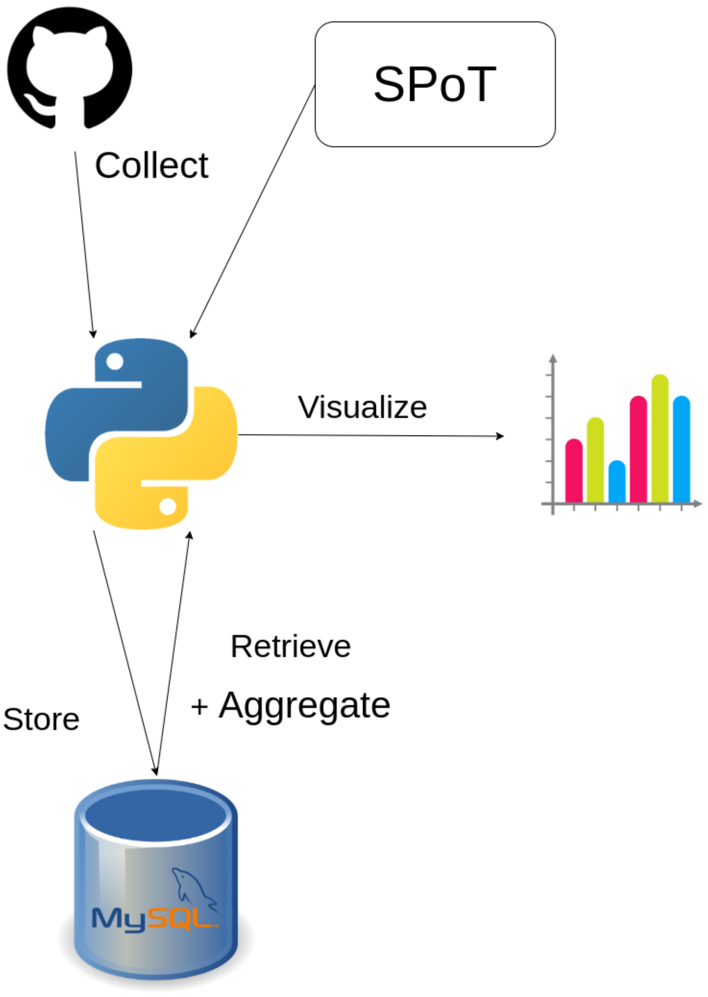


Figure 3. Methodology diagram

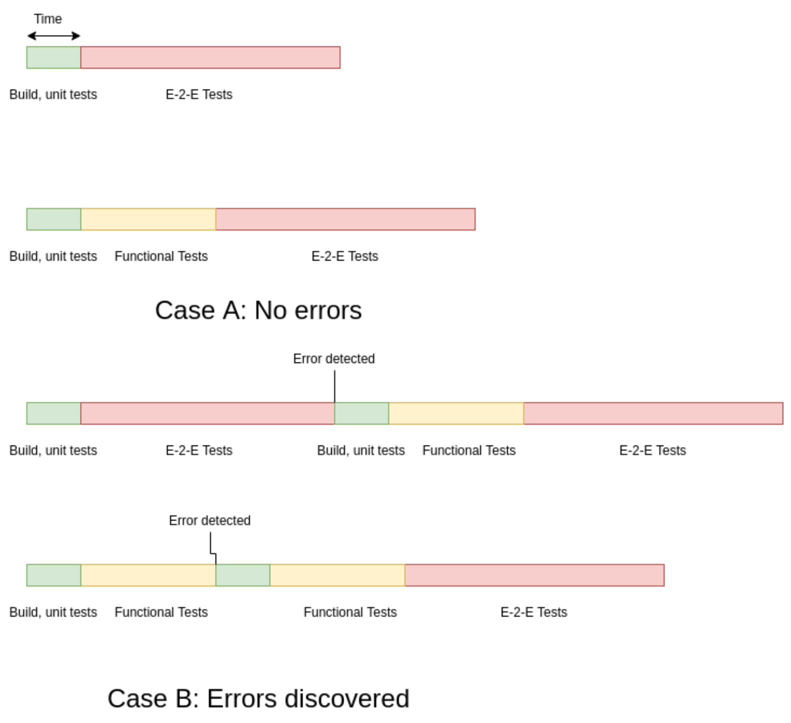


Figure 4. Approximation of the conjecture

request will be going live without any human supervision, and definitely with least amount of resources spent on it, since every deployment needs to run through an end-to-end check. However if it fails at the end-to-end stage, then developers most definitely will have to run the entire test suites, functional tests included, in order to better illustrate where the component might be failing, and once they identify the fault and queue the deployment, e-2-e tests need to be run again. While this is potentially legitimate improvement to the solution, difference is not expected to be too big, as seen on figure 4, However the option might still be explored.

Pull requests can be identified from the first part of the practical analysis, where A: Dependabot was the sole commiter in the PR, B: Dependabot needed supervision. E-2-E and Functional tests are manually going to be run and timed. This will determine how much time does the existing method win in the Case A - with no errors, and how much does it lose, in case B - with errors.

4 Implications of the Results

4.1 Analysis of Github Repositories

The data drawn from Github API was analyzed in three ways - The amount of lines of code that were changed per pull request, the amount of average number of commits in pull requests, and the amount of merged, unmerged and total Pull Requests made by the Dependabot.

In figure 5 we see an interesting trend, where the number of code, on average, has been decreasing in Pipedrive's repositories since the introduction of security tests. This can be down to old code being reviewed, and what is not needed, deleted. The possibility that it is some sort of mistake correction is small, since it does not make sense to correct git history through a PR, rather than directly rewriting the history - one leaves trace of potentially sensitive information, other does not. Overall, less code means less ways attackers can breach the system, therefore it is net positive in terms of security for Pipedrive.

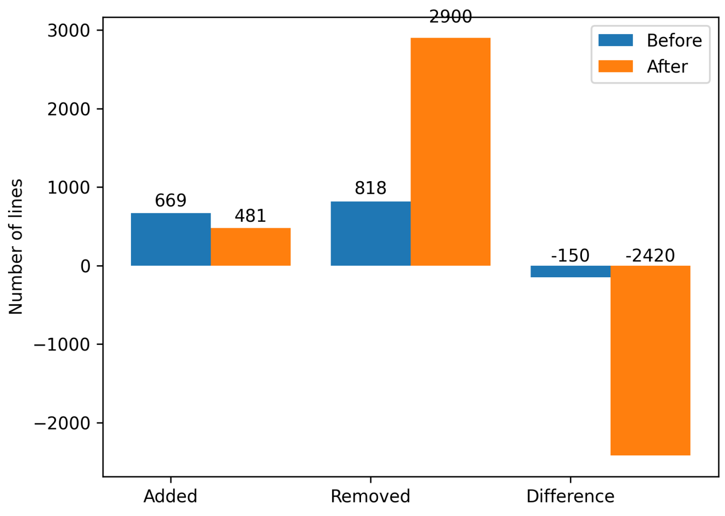


Figure 5. Average number of Lines Added/Removed before and after the introduction of security checks

Overall, from figure 6,it can be said that the number of commits has not changed too much - upon close investigation even the maximum number of commits is only a bit inflated in the older Pull Requests. There are only five more PRs with a commit number of 50 or more. This means that average effort - in terms of commits - hasn't changed. However, if coupled with the figure 5, the "effort" argument might start to break down, and it can be even said that "Pipedrive is getting better programmers", as some older programmers prefer to remove code from software to make it simpler, more elegant[26].

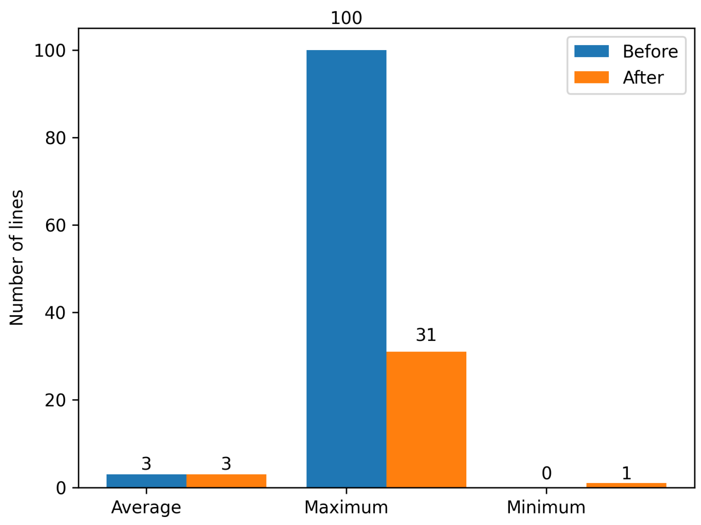


Figure 6. Average number of commits in a PR before and after the introduction of security checks

Most interesting for the next part is the comparison between merged and unmerged Dependabot Pull requests - out of all the dependabot PRs that were opened, only 41% were merged, and of the merged PRs, 58% had a single commit from Dependabot. The distribution of these PRs between pull requests based on their test coverage is shown in figure 7. While in the earlier chapters we've said that better code coverage implies more merged dependabot PRs, it is absolutely clear from this figure that in Pipedrive's case, this is not the case. In fact, the repositories that are in the highest percentages of code, merge less than one third of dependabot PRs.

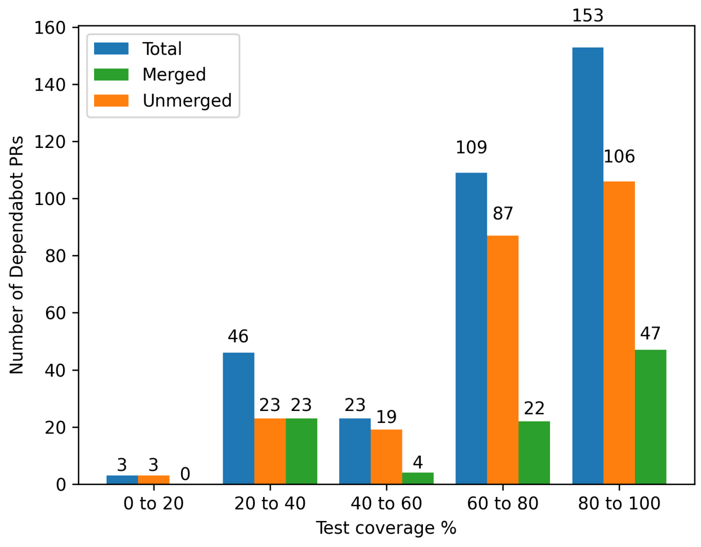


Figure 7. Distribution of Dependabot PRs

5 Summary

As can be seen from the research, improvements in security do not necessarily have to sacrifice usability or increase the effort bourne by the development teams - on average, number of commits is the same. On the contrary, security checks seemed to have improved the engineering quality in Pipedrive, e.g. removal of code. Meanwhile, the newly introduced Dependabot PR auto-merging will surely decrease vulnerabilities in dependencies.

Further studies can try to look into the correlation of overall dependency number vs number of commits and/or lines introduced to code, to verify the correlation that has shown itself in this paper. Further studies can focus on implementing more types of tests before E2E tests are going to be run in Pipedrive's infrastructure, to lessen the load on it, and to hopefully increase the overall throughput of it.

Bibliography

[1] Pipedrive. *About Pipedrive*. [Accessed: 12-05-2022]. URL: https://www.pipedrive.com/en/about.

[2] Sten Hankewitz. *Estonian-founded Pipedrive sells majority to an investment firm, becomes a unicorn*. [Accessed: 12-05-2022]. URL: https://estonianworld.com/business/estonian-founded-pipedrive-sells-majority-to-an-investment-firm-becomes-a-unicorn/.

[3] PagerDuty. *What is a Deployment Pipeline*. [Accessed: 15-05-2022]. URL: https://www.pagerduty.com/resources/learn/what-is-a-deployment-pipeline/.

[4] AWS - Amazon Web Services. *What are microservices*. [Accessed: 25-04-2022]. URL: https://aws.amazon.com/microservices/.

[5] Github. *About Dependabot security updates*. [Accessed: 12-05-2022]. URL: https://docs.github.com/en/code-security/dependabot/dependabot-security-updates/about-dependabot-security-updates.

[6] Snyk Limited. *Snyk Open source security management*. [Accessed: 12-05-2022]. URL: https://snyk.io/product/open-source-security-management/.

[7] Snyk Limited. *Snyk Open Source License Compliance Management*. [Accessed: 12-05-2022]. URL: https://snyk.io/product/open-source-license-compliance/.

[8] Snyk Limited. *Static Application Security Testing (SAST)*. [Accessed: 12-05-2022]. URL: https://snyk.io/learn/application-security/static-application-security-testing/.

[9] Aqua Security. *Trivy*. [Accessed: 12-05-2022]. URL: https://github.com/aquasecurity/trivy.

[10] Snyk Limited. *Snyk Container*. [Accessed: 12-05-2022]. URL: https://snyk.io/product/container-vulnerability-management/.

[11] Tom Preston-Werner. *Semantic Versioning*. [Accessed: 25-04-2022]. URL: https://semver.org/.

[12] ISO/IEC. *Information technology - Security techniques - Information security management systems - Requirements (ISO/IEC 27001 :2013,IDT)*. Switzerland, 2013.

[13] ISO/IEC. *ISO/IEC, "Information technology – Security techniques-Information security risk management"*. Switzerland, 2008.

[14] Tom Palmaers. "Implementing a Vulnerability Management Process". In: (). [Accessed: 24-04-2022]. URL: `https://www.sans.org/white-papers/34180/`.

[15] *Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration*. New York: Springer Science+Business Media, 2017. DOI: `10.1007/s10664-017-9521-5`.

[16] *How do APIs evolve? A story of refactoring*. Urbana-Champaign, Urbana, IL 61801, 201 N Goodwin Avenue, United States, 2005. DOI: `10.1002/smr.328`.

[17] *How to break an API: Cost negotiation and community values in three software ecosystems*. 2016. DOI: `10.1145/2950290.2950325`.

[18] *Historical and Impact Analysis of API Breaking Changes: A Large-Scale Study*. 2017. DOI: `10.1109/SANER.2017.7884616`.

[19] *Demo: Detecting Third-Party Library Problems with Combined Program Analysis*. 2021. DOI: `10.1145/3460120.3485351`.

[20] Gene Spafford. [Accessed: 25-04-2022]. URL: `http://www.cs.umsl.edu/~sanjiv/sys_sec/security/sys_insecure.html`.

[21] Jevgeni Demidov. "Our Custom-Built DevOps Tools Enable Us to Deploy Code in Production in Just Two Clicks!" In: (). [Accessed: 22-04-2022]. URL: `https://medium.com/pipedrive-engineering/our-custom-built-devops-tools-enable-us-to-deploy-code-in-production-in-just-two-clicks-d02e4fe215e4`.

[22] *Can we trust tests to automate dependency updates? A case study of Java Projects*. 2021. DOI: `10.1016/j.jss.2021.111097`.

[23] Saranga Komanduri et al. *Of Passwords and People: Measuring the Effect of Password-Composition Policies*. [Accessed: 25-04-2022]. URL: `https://users.ece.cmu.edu/~lbauer/papers/2011/chi2011-passwords.pdf`.

[24] et al. Eduardo Witter dos Santos. "Investigating the effectiveness of peer code review in distributed software development based on objective and subjective data". In: (2018). DOI: `10.1186/s40411-018-0058-0`.

[25] *On the Use of Dependabot Security Pull Requests*. 2021. DOI: `10.1109/MSR52588.2021.00037`.

[26] Pete Goodliffe. O'Reilly, 2014. ISBN: 978-1491905531.

Appendix 1 – Non-exclusive license for reproduction and publication of a graduation thesis[1]

I, Dachi Mshvidobadze

1. Grant Tallinn University of Technology free license (non-exclusive license) for my thesis "Case Study of Automated Vulnerability Management of Microservices at Pipedrive OÜ", supervised by Kaido Kikkas
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive license.
3. I confirm that granting the non-exclusive license does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

04.17.2022

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.