MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Implementation of Secure SW Development Lifecycle Framework

Master's Thesis

BC. KATEŘINA RAPČANOVÁ

Brno, Fall 2024

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Implementation of Secure SW Development Lifecycle Framework

Master's Thesis

BC. KATEŘINA RAPČANOVÁ

Advisor: RNDr. Josef Spurný, Ph.D.

Department of Computer Systems and Communications

Brno, Fall 2024


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Kateřina Rapčanová

Advisor: RNDr. Josef Spurný, Ph.D.

Acknowledgements

I would like to thank my advisor, RNDr. Josef Spurný, Ph.D., for answering all my questions and for all his guidance. A big thanks also to the advisor from the company, as well as all the colleagues, who were always ready to help me and gave me even more insight into the company. Last but definitely not least, a huge thank you to my partner for trying to keep me motivated and for making my life outside of the university world so much easier. That also goes for my friends and family, thank you.

Abstract

The aim of this diploma thesis was to choose an appropriate Secure SW Development Lifecycle (SSDLC) framework and implement it in a selected company.

Five SSDLC frameworks were thoroughly analyzed and compared based on selected focus points and metrics, as well as the security analysis of the company's SW development lifecycle (SDLC).

Based on the comparison results, the NIST: Secure Software Development Framework (SSDF), together with Microsoft Security Development Lifecycle (MSDL), were chosen as most suitable for the company from the five analyzed frameworks.

Two new policies, the Secure Software Development Lifecycle Policy and Secure Coding Policy, were developed to implement the SSDF and MSDL into the company. Each phase of the company's SDLC was improved by the corresponding practices from the selected frameworks.

The thesis concludes with a discussion of the challenges that emerged during the implementation and the possible further improvements to the company's SDLC.

Keywords

Secure Software Development Lifecycle (SSDLC), Software Development Lifecycle (SDLC), secure development, application security, software security, MSDL, SSDF

Contents

Introduction **1**

1 Chosen SSDLC Frameworks and Methodologies **3**
1.1 NIST: Secure Software Development Framework (SSDF) 3
1.2 Microsoft Security Development Lifecycle (MSDL) . . 4
1.3 Securosis: Putting Security Into DevOps 9
1.4 Juniper Networks Secure Development Lifecycle (JNSDL) 13
1.5 Cisco Secure Development Lifecycle (CSDL) 16

2 Introduction to the Company **19**
2.1 Development Lifecycle 20
2.1.1 SDLC of a Single Product Change 21

3 Security Analysis of the SDLC **24**
3.1 Planning 24
3.2 Analysis 25
3.3 Designing 25
3.4 Implementation 25
3.5 Testing 26
3.6 Deployment and Maintenance 27
3.7 Security Strengths 28
3.8 Main Security Flaws 30

4 Choosing the Best Framework **35**
4.1 Focus Points and Metrics in General 35
4.2 Comparison Based on the Found Security Flaws 37
4.3 Chosen Framework with Reasoning 40

5 Implementing the NIST SSDF and MSDL **41**
5.1 Preparation for the Implementation 41
5.2 Improved SDLC 44
5.2.1 Planning 44
5.2.2 Analysis 45
5.2.3 Designing 46
5.2.4 Implementation 47
5.2.5 Testing 52

5.2.6 Deployment 55
5.2.7 Remediation and Maintenance 56

6 Found Hindrances during the Implementation **59**

7 Possible Future Security Improvements **62**

8 Conclusion **64**

A Electronic Attachments **66**

Bibliography **67**

List of Tables

4.1 Comparison of chosen methodologies based on the focus points 37
4.2 Comparison of chosen methodologies based on the security flaws 38

List of Figures

1.1 SSDF practices [5] 4
1.2 SDLC according to MSDL [10] 5
1.3 Toolchain proposed by Securosis [21] 12
1.4 JNSDL practices [22] 13
1.5 CSDL overview [23] 17

2.1 Organizational structure 19
2.2 Developers' activities 20

5.1 Practices for the phase Preparation 42
5.2 Practices for the phase Planning 44
5.3 Practices for the phase Analysis 45
5.4 Practices for the phase Designing 47
5.5 Practices for the phase Implementation – Environment . . 49
5.6 Practices for the phase Implementation – Implementation 51
5.7 Practices for the phase Implementation – Code review . . 52
5.8 Practices for the phase Testing 54
5.9 Practices for the phase Deployment 55
5.10 Practices for the phase Remediation and Maintenance . . 57

Introduction

When designing a product, be it software or hardware, there should be a clear development lifecycle with given phases. However, not everyone considers security when developing a product, at least not at every stage of the development. In software development, the cycle usually consists of the following phases: planning and analysis, design, development, testing, deployment, and maintenance [1]. The question now is where should developers consider the security of their product? And an even more important question is why should they care about security?

The main goal of securing the SDLC is to ensure the confidentiality, integrity, and availability of software components and data, as well as the product as a whole. With a secure SDLC, end users of the software should be assured that there are no exploitable vulnerabilities left and can use it without fear of data loss [2].

There are many approaches to securing the SDLC. In a systematic mapping study conducted in 2020, more than 50 such approaches were found. Mostly, static analysis and dynamic analysis of a code are used to secure the resulting products [3]. However, such an approach focused only on one part of the SDLC is insufficient, as all the phases should be secured. The security weaknesses should be discovered right at the beginning, at the stage of setting requirements. If that is not the case, the cost of fixes increases the later the weakness is found [4]. Therefore, it is best to incorporate security into each lifecycle stage so that the SDLC turns into the Secure Software Development Lifecycle (SSDLC).

When implementing SSDLC in a company, one can follow several publicly available frameworks and methodologies. They usually do not address specific measures or tools the company should incorporate but provide guidelines and best practices each company can introduce to their SDLC as they see fit. Therefore, the company must first evaluate its security level and requirements from such a framework. Only then can the responsible team decide which framework works best and start with its implementation.

For this thesis, five such frameworks and methodologies are chosen, described and compared based on chosen metrics and the com-

pany's security analysis. Then, the most suitable framework will be chosen and implemented into the company's SDLC, making it closer to becoming an SSDLC.

1 Chosen SSDLC Frameworks and Methodologies

The company is using agile development, and therefore, there will not be any frameworks that are specifically published for different development styles. On the other hand, many frameworks and methodologies are designed in a way that they can be used for any development strategy, and some of these are mentioned below.

1.1 NIST: Secure Software Development Framework (SSDF)

The framework was first published in April 2020. The current version of this framework is 1.1, which was published in February 2022. There are four groups of practices in this framework. First is **Prepare the Organization** (**PO**), then **Protect the Software** (**PS**), **Produce Well-Secured Software** (**PW**), and lastly, **Respond to Vulnerabilities** (**RV**). Furthermore, each specific practice has four parts: **Practice, Task, Notional Implementation Example, and Reference**. The practices are often based on documents by organizations such as OWASP[1], BSA[2], and SAFECode[3]. The representation of the practices can be seen in the figure 1.1.

The SSDF is not necessarily meant to be used as a whole. The practices are outcome-based, so the organization can compare their current outcomes to the ones outlined here to see what needs to be worked on. The organization has to further decide, based on its resources, which practices it should implement. However, that said, there are some practices that depend on other, more fundamental ones. All in all, the framework is not to be used as a checklist but to provide the basis for a risk-oriented approach in the company during the whole SDLC. Although this framework is not structured around the different SDLC phases, the practices can easily be fitted to them [6].

1. The Open Worldwide Application Security Project, `https://owasp.org/`
2. Business Software Alliance, `https://www.bsa.org/`
3. `https://safecode.org/`

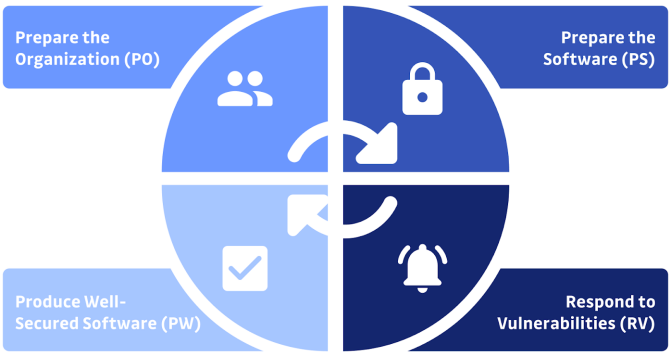


Figure 1.1: SSDF practices [5]

When looking more closely at the groups of practices, it might really seem like the practices themselves are just a checklist. However, because it is only a framework, no clear way to implement the practice is given. For this purpose, there are examples for each practice task to give the reader a sense of how to implement it. By PO, there are five practices divided into thirteen tasks. PS consists of three practices with four tasks. PW is the most extensive group, with eight practices and sixteen tasks. RV has three practices and nine tasks [7].

1.2 Microsoft Security Development Lifecycle (MSDL)

The first version of this framework was released in 2004. However, the first public release was in 2008. To this day, the sixth version has been released to the public. The current version was published in May 2024.

Since Microsoft officially uses the framework for DevSecOps (Development, Security, and Operations), it should be possible to use it to adapt DevSecOps to any other company that also uses DevOps. Microsoft also refers to their many tools and resources in each practice in the MSDL, so the reader can implement it using the tools they might be already using or can purchase [8].

The best practices of the MSDL are also described in a book [9]. The practices are a bit different from the current version published on

the Microsoft website; however, they can serve as a great additional source if a company chooses MSDL as their SSDLC framework.

The current MSDL outlines ten best practices that are described below and together integrate the security into the SDLC as shown in the figure 1.2.

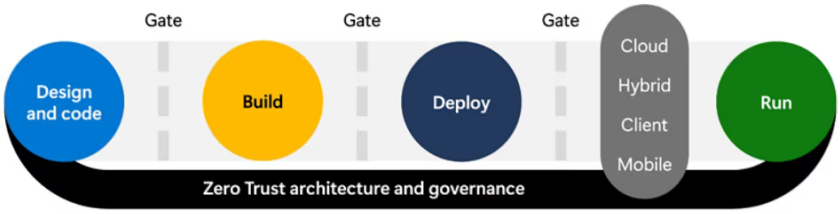


Figure 1.2: SDLC according to MSDL [10]

Establish security standards, metrics, and governance

This practice is about defining the security and compliance baseline and continually updating it to reflect all changes, be it in the regulations, the software's functionality, or new possible threats. It also stresses the need to communicate these baselines to the team members and to make sure they follow them. Furthermore, there is a need for exception handling in a proper manner. This practice consists of four steps: Identify Required Standards, Define Security Requirements, Define metrics and compliance reporting, Create a Security Exception Process [11].

Require use of proven security features, languages, and frameworks

The development teams should only use security solutions that are already established and proven because it is not recommended to

invent new ones, as this usually ends in even higher security risk. There should be a list of approved tools (also provided by Microsoft as SDLC Tools), and all should be regularly updated to the newest version as fast as possible. The practice has five steps: Identity, AI safety and security, Data protection: Securing content used in apps, Logging and telemetry, Use approved tools [12].

Perform security design review and threat modeling

Given the constant risk of adversaries being able to exploit system vulnerabilities, it is necessary to perform threat modeling and security design reviews. The developers should then be able to mitigate possible threats during the development process. The shift to the mindset of "how our products might be abused" instead of only focusing on their functionality is crucial here. The practice has six steps: Identify use cases, scenarios, and assets; Create an architecture overview; Identify the threats; Identify and track mitigations; Communicate threat models to key stakeholders; Threat Modeling resources [13].

Define and use cryptography standards

Once the attacker is able to find a weakness in chosen cryptography practices, the whole system becomes insecure. Therefore, it is important to choose the correct solution from the beginning, and that goes both for securing data at rest and in transport. This, of course, depends on the data itself and how much it needs to be protected. Since not every developer is a cybersecurity expert as well, there must be guidelines as to what cryptography practices to use and how. There are four steps to the practice: Encrypt data in transit and at rest, Post-Quantum Cryptography (PQC), Cryptographic agility, Encryption key and certificate management and rotation [14].

Secure the software supply chain

Since many software nowadays are built using third-party components (libraries, extensions, etc.), if one component gets compromised, the whole software is no longer secure. Before adopting such third-party solutions, one should do a thorough analysis, especially if it is a high-

risk component. The consumption of OSS should be, therefore, defined in a uniform process and standardized for each developer. The steps in this practice are: Establish a secure open source software ingestion process, Understand dependencies in your environment, Produce software bill of materials (SBOM), Signing and attest artefacts [15].

Secure the engineering environment

It is extremely important to secure the operational environment against attackers that could abuse or even add vulnerabilities and back doors. Not even user accounts and non-production environments are safe and need to be secured just like the code itself. Security logging, Just-In-Time, Just-Enough-Access (JIT/JEA) and least privilege access policies should be implemented. This should be accomplished by the following six steps: Take a Zero Trust approach, Disallow direct commits to production branches, Implement privileged access workstations, Provide Secure virtual workstations, Implement GitHub Codespaces, and Secure deployment environments [16].

Perform security testing

In order to see potential risks and be able to decide on secure processes during the development process, it is essential to perform security testing. It is one of the most common and usually the first step in making the SDLC more secure. The testing can be done manually or automatically, and usually, more than one approach is used. The steps for this practice are basically the tests that should be done: Implement Static Analysis Security testing (SAST), Implement dynamic analysis security testing (DAST), Red/blue team exercises, Application penetration testing, Perform continuous security testing and measurement, Perform triggered/cadence-based security testing, Run a bug bounty program [17].

Ensure operational platform security

If an attacker compromises the security of the infrastructure hosting the workload, they can take over the data and workload or even launch an attack on other systems. Each workload has to be secured, and that

does not only apply to the production workloads but also to the whole infrastructure. This practice is most detailed so far with ten steps: Enforce multifactor authentication, Protect administrative accounts, Implement security baselines, Create isolation layers, Use confidential compute, Reduce the attack surface, Perform platform penetration testing, Implement operational terminals: device security, Scheduled maintenance and automated patching cycles, Protect against DDoS attacks [18].

Implement security monitoring and response

This practice stresses the need to monitor both sides – known vulnerabilities as well as signs of a possible active attack. This is important for vulnerability management and ensuring that the company can recover from attacks. Usually, security operations manage active incidents. In contrast to the last practice, this one offers only two steps: Proactively detect and address threats, Establish a standard incident response process [19].

Provide security training

Every person in a company whose decisions somehow influence the security of the product needs to be properly trained. They need to understand the why, what, and how when it comes to addressing security risks, as this can help them not see security only as an annoyance. The fact is that during an SDLC, almost every involved person's decisions can have an impact on the overall security. That said, the programmers and other technical employees are rarely experts on cybersecurity and need, therefore, to undergo training not only regarding the technical parts but also the conceptual ones. Threat modeling is one of the most important trainings for engineering teams so that the products can be Secure by Design. On the other hand, not everyone needs to be a security expert, so it should be decided who needs to know only the basics and who needs more thorough training. In all cases, the training should not be a one-time thing but a continuous and repeated process to ensure awareness in the dynamic world of security attacks and vulnerabilities. There are no steps for this practice. However, there

is a list of resources on GitHub or Microsoft Learn to help with the security training process [20].

1.3 Securosis: Putting Security Into DevOps

Securosis as a company has published many white papers that could be used to secure an SDLC. The most interesting ones with consideration of the company are Secure Agile Development [4], Putting Security Into DevOps [5] and Building a Web Application Security Program [6].

They are all a bit older than the other frameworks and methodologies. However, that does not necessarily take away from their relevance, especially since the described processes and techniques are still widely used today. For this thesis, the white paper **Putting Security Into DevOps** [21] was chosen to be compared with the other frameworks, especially its parts *Mapping to an SDLC*, *Security Tools for DevOps*, and *Security's Role In DevOps*, mainly because the described company uses the DevOps approach. Firstly, it is explained in the paper that even though most developers would not consider SDLC as a development process for DevOps, the phases are not really dependent on the development process and can be easily adapted into DevOps as well.

During the **Define** phase, there are three important parts to consider. The first one is *Operational Standards*, where the importance of continuous improvement of an operational environment is stressed. Configuration management, build automation and the way patches are incorporated, to name a few, should be accounted for. During the later development phase, it is operated with these standards in order to shape the stories. Then come *Security Requirements* with the reminder to run security tests before the code is accepted. The baseline should be set (e.g. OWASP Top Ten vulnerabilities to be mitigated), and appropriate tools should be chosen to help abide by the baseline for old and new functionalities. Some tests can be done even post-production

4. https://cdn.securosis.com/assets/library/reports/SecureAgileDevelopment_Nov2014_FINAL.pdf
5. https://cdn.securosis.com/assets/library/reports/Security_Into_DevOps_Final.pdf
6. https://cdn.securosis.com/assets/library/reports/WebAppSec_Programv1.pdf

so as not to hold up the release. Last is *Monitoring and Metrics*. Since the improvements are made iteratively with each release, the metrics and their monitoring should make clear what else needs to be fixed or improved, what is fine just as is, etc. The metrics also evolve with time, but it is important to have some basic set right from the beginning.

In the phase of **Design**, the first mentioned thing is *Secure Design/Architecture*. DevOps greatly enhances security by enabling automated patching and configuration, reducing the need for manual administrative access to production servers, which removes a common security risk. By leveraging automation, such as automated certificate injection and patching, teams can preemptively address errors, making systems more resilient and secure without manual intervention. However, the deployments must be specifically designed to be able to use all the advantages. To *Secure the Deployment Pipeline*, even the usually less secured development and test environments should be more secured in DevOps, as they are much more connected to the production itself. The build servers and code management especially need stricter access management and monitoring. The tracking of who made any changes also needs to be provided. Lastly, the importance of a *Threat Model* is presented. Since threat modeling is one of the most important security exercises, it should also be incorporated into the DevOps approach. The developers should be informed about possible attacks based on the threats and try to mitigate them with unit tests. The modeling itself can also be done later during the development phase itself; however, most often, it is done during this phase.

During the **Develop** phase, planning and building of the tests is done upfront before the new features are developed. In each build, it is possible to reuse already established tools and infrastructure, so the process is more efficient. The operations team is also an important part of the development process in DevOps, mainly providing the build scripts for the development, testing, and production infrastructure. With the help of automatization, IT issues such as undocumented and delayed patches should be reduced, and a more secure baseline for systems should be established. However, most companies have not yet fully incorporated automation. No code that is broken or untested should be released. Policies should be written in functional scripts, and on top of that, unit and functional tests should enforce the higher security requirements. To further ensure security, at least one person

(preferably one who is interested in security topics) in every team should be trained on security basics. With that, in-house experts can be created.

Then comes the **Test** phase, where it is important to strive for failure and parallelize security testing. In DevOps, the principles are completely different, and so, for example, quality assurance tries to break the application rather than get the code to pass its requirements. The new principles raise the security baseline, so the minimal quality and robustness of the code are higher than before. This helps find ways to break an application before its release and before the attacker can. Furthermore, due to some tests running longer than the development itself, different security tests are run parallel, such as code scanners and unit tests. The results are then examined by the developers. The line between tests for efficiency vs. speed has to be balanced.

The last phase is **Deployment**, where it has to be decided between manual vs. automated deployment. Since the rollback is usually much harder than the release itself, most companies still do not rely on a fully automated deployment that would allow them to release new versions every day. However, some level of automatization is always better as the personnel can then focus on testing and monitoring. When talking about deployment and rollback, development teams still perform "smoke tests" to verify that a code works in a live environment as well as it did in pre-deployment tests. The tests are now more evolved, and the three most common ones are Blue-Green or Red-Black deployment, where old and new code is run parallel, and in case of an error, the load balancers go back to the old code; canary testing, the new code is tested by a small group of people and in case of an error (the canary dies), the new code has to be fixed, and then the test is done again; last one is feature tagging, which allows teams to enable or disable specific features in real time if problems occur. Production security tests are also a big part of deployment, as the applications can usually work even when the security controls fail. The validation of the production environment by penetration testers, code scanners, etc., needs to be done to make sure the critical security controls are in place and functional.

The part of the paper that describes the **Security Tools for DevOps** firstly focuses on where to perform tests, then how to build a toolchain and lastly on prioritization and possible risks. The tests mentioned are

unit testing, security regression tests, production runtime monitoring and managed releases. It is important to incorporate all of them side by side with functional and regression tests throughout the development and deployment phases. The toolchain shows different testing techniques and their place in the DevOps process. The toolchain can be seen in the figure 1.3. The different testing tools that are mentioned are SAST, DAST, fuzzing, manual code review, vulnerability analysis, version controls, and runtime protection. When talking about the priorities and risks, one has to realize that it is not the technical part of the integration of security findings into bug-tracking systems that is difficult but what to do with the obtained data. It must be decided what risk is just a false positive, if the vulnerability can actually pose a risk, etc. Therefore, both Dev and Ops have to be included in security management.

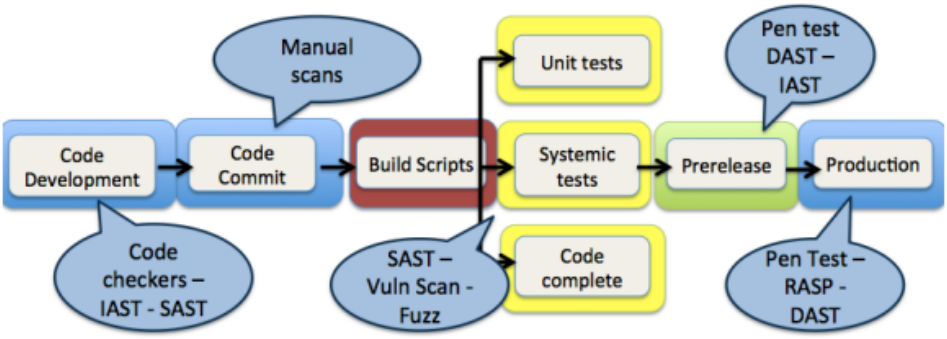


Figure 1.3: Toolchain proposed by Securosis [21]

In **Security's Role In DevOps**, there are the responsibilities of a Chief Information Security Officer (CISO), but also the way a CISO supports DevOps described. The security personnel can sometimes struggle with how to work with the development team during the SDLC. These guidelines should help clear that up. Security should be a part of the DevOps framework, and the personnel should try to improve the SDLC without great difficulties for both sides. The main responsibilities of a CISO should be to learn the DevOps process (with help from someone from the development team) and to learn to be agile themselves (to fit into DevOps, not the other way

around). To support DevOps, a CISO can educate developers, help them understand the threats, help with writing tests, etc.

1.4 Juniper Networks Secure Development Lifecycle (JNSDL)

This framework [22] introduces six practices that should help improve product security. The practices of this framework are also a part of a more complex Juniper's overall Product Development Lifecycle (PDL) methodology. They add a layer to the PDL or a different SDLC and are more lightweight, as not really meant to be used by themselves. The practices should address the principles of Confidentiality, Integrity, Availability, Non-repudiation, and Authentication. They are mostly oriented toward the Juniper employees themselves. However, it is possible for outsiders to also get inspiration from them.

The practices are shown in the figure 1.4 and described in more detail below.



Figure 1.4: JNSDL practices [22]

Secure Coding Training

Every developer should undergo secure coding training in order to be able to write secure code. There are trainings for many programming languages, so every developer is trained for the one they will be using. The training should mainly cover different topics of secure coding, secure design, and privacy. Some of the topics are, for example, Buffer overflows, Input and output validation, Cross-site scripting, Principle of least privilege, Security testing, Code review, Static and dynamic analysis, and Privacy management. It is every involved person's responsibility to conform to security training. That also applies to managers and IT personnel. Therefore, extra training is also available to cover the basics of security testing principles, application security, and IT security.

Security Considerations in Design

The importance of incorporating security assessment into the early phases of the SDLC is again emphasized here. The developers have to consider security during the design of the software, and that is done by evaluating vulnerabilities, thinking about how a vulnerability could be used, and trying to mitigate the vulnerabilities by designing a better system. It is important not only to look at the software itself but also how it will be operated in different environments. The security vulnerabilities could be different for each scenario, and it is important to evaluate all possible ones. The security of various features can be assessed through activities like vulnerability assessments, which utilize industry-standard information and resources such as RFCs. Additionally, security posture evaluations can be conducted based on the product's software functional specifications, and security feedback can be incorporated into the product's planning and design stages.

Threat Modeling

Since developers do not always find every possible vulnerability in their product and also do not think as an attacker who will use every possible means to exploit any vulnerabilities, it is important to understand the threats as well to build a truly secure product. Threat modeling helps the developers identify and hopefully mitigate threats

not only to the product itself but also to other components that interact with the product. It should also show which threats are more likely and could be more destructive than others. The inputs for this practice can also come from sources other than the threat model itself, such as other SDLC practices or Common Vulnerabilities and Exposures (CVE) identifiers.

Penetration Testing

Once the threat model and security posture of the product is defined, the security risks also need to be tested and validated. Penetration testing is one of the most common methods to do that. By using ethical hacking, the tester tries to really break the product in a way a real attacker might try to. With the use of manual and automated tests, new vulnerabilities or new ways of exploiting new vulnerabilities can be found. The tests depend on what was specified as an attack surface in the threat model. The found security issues should be mitigated before the product release and can serve as a security baseline for future product development and testing.

Release Security Review

Before the product can be released, its security posture needs to be examined. In this practice, the results from threat modeling analysis, penetration testing, and previous product security reviews are evaluated. It is also considered if the secure coding training requirements and functional specifications were met, recommendations for product enhancement were incorporated, and bugs reviewed. The security team can decide based on all those inputs if the product is ready to be released or if there is a need to fix some issues first.

Incident Response Plan

Even if a product is released without any vulnerabilities (this is, however, never the case), vulnerabilities can still be found over time. Therefore, there needs to be an incident response plan that covers responses to new security issues and also outlines how to communicate the issues and their fixes to customers. Juniper has a dedicated Security

Incident Response Team (SIRT) that works with the plan. The plan relies on existing SIRT methodologies, such as having established communication channels for incident resolution, looking for potential vulnerabilities in different sources, having dedicated security experts for incident resolution, evaluating and scoring security bugs, or sharing information about found threats and vulnerabilities with peers to mitigate their spread.

All in all, this framework is, as already mentioned, mainly focused on the developers of Juniper systems. Maybe that is also the reason that it is so vague and short (only a ten-page document). It is possible that a more detailed version is accessible to Juniper employees, and this version is mostly for public information and usage.

1.5 Cisco Secure Development Lifecycle (CSDL)

Like with the Juniper framework, the CSDL is mainly focused on Cisco products and how they are developed. It is very short and has very few details. The CSDL is an always-evolving methodology that can be incorporated into every development stage. The core processes are Plan, Develop, Validate, Launch, Operate, and Monitor. They copy the SDLC itself, and for each process, there are specific steps that should be done. The CSDL focuses on secure-by-design philosophy throughout the whole system lifecycle, and its overview can be seen in figure 1.5.

During the **Plan** process, the steps that are unnecessary for incorporating security and privacy information in the technology right from the beginning are done. These range from gap analysis to risk assessment, threat modeling, and privacy assessment. Cloud security, according to industry certifications, also has to be considered during this process. Privacy controls are continually re-evaluated in order to stay compliant with local requirements.

In the **Develop** process, the practices of Cisco developers are described. They follow many secure coding best practices, and when necessary, they use hardening technologies like object size checking (OSC), XSPACE, and address space layout randomization (ASLR). Other than that, repositories to store code are chosen based on their security and restricted source control. The developers can perform



Figure 1.5: CSDL overview [23]

a peer review of others, helping with knowledge-sharing and error reduction. They are also using common security modules that were previously reviewed by Cisco and help the product be more resistant to threats – for example, CiscoSSL on top of OpenSSL (CiscoSSL can be patched independently of OpenSSL, saving precious time when a vulnerability is discovered). The products are checked for vulnerabilities by SAST tools, and new software releases are also scanned for security issues. Lastly, employees are trained in order to increase their knowledge of security and privacy. If needed, the security education can be multi-leveled.

The **Validate** process is a process of testing the products for finding and possibly mitigating security defects. The testing can be done using open-source tools, protocol tests, and other application test methods. Possible methods include vulnerability and penetration testing (with Red Team attack simulations or third-party penetration testing when necessary), making sure that all third-party software from the software bill of materials (BOM) is compliant and without vulnerabilities

and also privacy control validation, including controls such as data classification and encryption when applicable.

During the **Launch** process, the pre-launch criteria have to be met in order to prevent security risks and to properly prepare the product for customer use. The Cisco Product Security Incident Response Team (PSIRT) can postpone the release if a critical security defect is discovered.

After the successful launch of the product, the process **Operate** comes next. To ensure the continuous security of the product, regular maintenance releases according to the CSDL for on-premises products are published. For cloud products, the process is even more complex with automated vulnerability testing, continual hardening and identity management, to name a few. Another important part, especially for cloud products, is maintaining privacy controls. They need to be designed by default to comply with legal requirements.

The last process, **Monitor**, is especially important because of the constant possibility of new threats. The CSIRT constantly monitors and evaluates all logs, services and data centres. Different tools are also used to detect and respond to any new threats. Other than security, the value chain of the products throughout their lifecycle is monitored and improved if need be [24].

2 Introduction to the Company

The chosen company is a software company with agile development that also incorporates Information Technology Infrastructure Library (ITIL) practices into its processes. The company focuses on IT service management software that can be incorporated with Microsoft technologies and applications. With around 40 employees, the company is a small business. However, even though it is a small company, it has over 25 years of experience in the IT market.

There are five departments there, and with its two branches, the company is divided into the departments: Business, Implementation, Technical, and HR are in one branch, whereas Development, IT and Marketing are based in the other. Notable here is that the sales and development directors are also the company owners. The security manager with the security team stands alone in the structure to signify their independence and importance throughout the whole company, as can be seen in the figure 2.1. The secure structure of the company and especially the management also belongs to the requirements for developing secure software [25], so it is important to consider where the security team and especially its manager stand in the company.

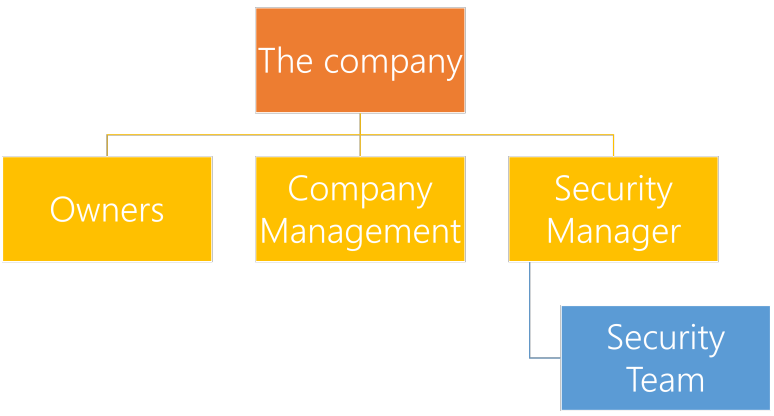


Figure 2.1: Organizational structure

2.1 Development Lifecycle

The company uses an agile development approach. The developers are divided into two teams – team A is focused on the front end, and team B on the back end. Each team has its own scrum team leader, and there is also a scrum master for both teams. Each developer has their time divided between different activities. These are divided as shown in chart 2.2.

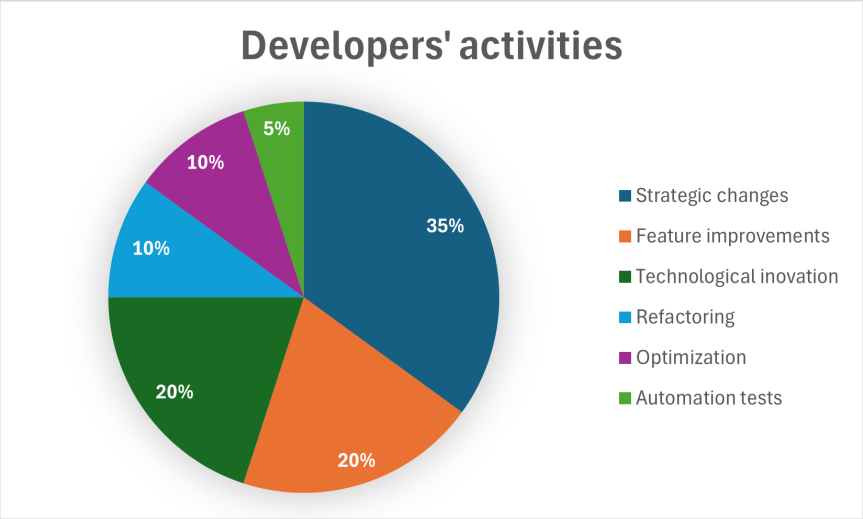


Figure 2.2: Developers' activities

The following description of the SDLC is taken from the processes and internal documents made specifically for it. The development of a big product version is divided into five stages. The first one is planning and takes up to two months. Next, there are four stages of the development itself. In each of the stages, there are the following phases: preparation, planning, development, and evaluation. These are repetitively performed and together sum up one sprint.

Preparation – In this phase, everything is prepped so that the next phases can run smoothly. That includes scheduling meetings, preparing folders and templates, etc.

Planning – The whole plan for the stage is set up at the scheduled planning meetings. Ideas from the customers, product road map, and product vision are all used as sources for what to include in the plan.

At this phase, consultants are also called to choose which changes in the product they want to oversee to make sure that the changes make sense from the customers' perspective. During the initial planning phase, only the most senior developers are present.

Development – In this phase, the changes are developed. Each change to the product has its own lifecycle, which will be described later. However, this phase is not further specified.

Evaluation – The whole stage is evaluated in retrospect. Questions such as "Did it meet expectations?", "What went well, and will we repeat it?" and "What will we do differently next time?" are asked and answered in order to improve in the next stages.

2.1.1 SDLC of a Single Product Change

As mentioned above, the development phase is not really further specified. However, a process is in place for each planned change being made to the product.

Classification

The main developers automatically get assigned the ticket for the change when it is first submitted. They have to fill out fields such as priority, target version of the product, workload, who will be the consultant, phase of the road map, issues with the current solution, etc.

Analysis

The analysis is done by a product management team, or if a technical analysis is really necessary, it is done by a developer who will develop the change. They have to further work with a consultant on the analysis and analyze the current state and the proposed change to estimate or refine the workload needed. The goal of this stage is also to create a request for any specific incremental changes to products if they are significant changes. When the analysis is done, it is sent to the Change Advisory Board (CAB) for approval.

Design

The developer, together with the consultant, prepares a design for the change. The CAB for the change consists of different people based on the character of the change. For example, if the change is based on an idea from a customer, they and all subscribers of the idea are also a part of the CAB team (however, the design is shortened and simplified when sent to these CAB members).

Coding

In this phase, the coding itself happens. The developer should no longer change the design without consulting the CAB team first. The coding happens on a separate branch of code, and first, when it is approved, it can be committed to the branch with the product's target version. In this phase, the developers should also decide whether there is a need for new automated tests. After finishing writing the code, they need to send their change to another developer for a code review. Only if the code review is successful can the developer move the ticket with the given change to the next phase. To help decide whether the change is done, the developers have a detailed Definition of Done (DoD) document, which they consult before sending the code for code review.

Documentation

When moving to this phase, a new ticket is created, and the person responsible for the user guide can start adding the new changes to the user guide. They can find all the information they need to put there in the design document.

Code review

A part of code review is, of course, checking for errors in the code itself, but also making sure that there are enough guidelines for the testers to be able to test the new change thoroughly. If there is any problem, the developer returns the ticket for correction to the original developer.

Testing

The testing should primarily be done based on the developer's notes. However, the testers should also think and test the change like regular users would and try all the possible scenarios the developer might have forgotten to mention. This includes some marginal cases, such as using special symbols, etc. The testers test the changes on the given target versions – builds. When the change has been successfully tested (works as should and no other errors have been discovered), it is deployed to the newest big version of the product. In case the testing is not successful, the testers return it to the developers for correction.

Returned for correction

If a problem is found during code review or the testing phase, the ticket is returned for correction to the original developer of the change. When everything is fixed, the ticked is returned either for another code review or straight to the testing phase. That depends on the extent of the correction.

3 Security Analysis of the SDLC

The goal of this chapter is to analyze what security practices are being used during the SDLC phases in the company and what is missing. As many frameworks and methodologies suggest, there have to be specific security policies for different phases of the SDLC and even within the phases. In general, it can be said that in each phase, the security of the software must be taken into account to prevent vulnerabilities in the next phases, as the cost of error corrections is higher in the latter phase, as already mentioned before.

For each phase, the question "How is security assessed during the phase?" is asked and then different metrics for each phase can be set [26]. However, since the company is just at the beginning of implementing SSDLC, these metrics can perhaps better serve in later iterations.

For the sake of keeping the SDLC as usually presented, the stages for the whole development are merged with the phases during the development of a single change. Therefore, the phases of Coding, Documentation, Code Review and possible Returned for correction are together under the Implementation.

3.1 Planning

Planning is mainly the domain of the Product Management (PM) team. They decide what will be solved in the next stages. As planning is mostly done only on a high level and a small group of people is present, the security of the final product or change to it is not really discussed here. Furthermore, the PM team is not composed of developers or other specifically technically able people, so they might not always see the possible security vulnerabilities or implications in the proposed changes. The only time the security is already considered during this stage is if the proposed change is blatantly endangering the safety of the product.

3.2 Analysis

During the analysis, security should already be discussed. It depends on the type of change and who does the analysis. Most analyses are done by the PM team, especially in case of a non-technical change. When the change is technical and above the technical skills of the PM team, the developer does the analysis. Since they should know the code, they should already think about the security implications of the change. That said, most analyses are still done by people who are mostly not developers and cannot see well what the change will influence in the code itself.

3.3 Designing

This is the stage where security should be considered from all possible angles. It should also be considered how to remediate possible new vulnerabilities caused in the future by this change, even if it is not a risk now (usually possible SQL injection, etc.). The developer, together with a consultant, discusses the design of the change, the effect it will have on the existing software, and what could go wrong when operating by the customer. Here, the responsible person for the change is set, and it is up to them to design the change so that it does not impose a new security vulnerability or threat. Even a recent newcomer can be in charge of a design and so of the whole change.

3.4 Implementation

During the **implementation**, the developers consider the security implications of their change. They do not use special extensions or code checkers but are aware of the most common security mistakes and try to avoid them. Once the change has been implemented, the developer checks the DoD. It is also clearly stated there that they should make sure there are no new vulnerabilities in the code. The developer also sends merge requests to the branch. This process is not controlled by a devoted developer, so basically anyone can approve a merge request, even the requester themself. During the phase of **code review**, the security is also checked again. The code review can be

done by any developer, even a junior developer. This is so that the newcomers can observe and learn from the more skilled developers. If the change is **returned for correction** because the testing failed, the code review is not always performed again before sending it back to testers.

Environment

Each developer sets up their coding environment. They usually use their work laptop. However, they are free to use a virtual machine (most do not). When there are temporary developers, they use their own devices that are not managed by the company. There is a list of recommended extensions to use, but these do not usually have to do with secure coding. Licensing of used programs and tools is controlled. Furthermore, there is a lot of third-party software in the product itself, such as libraries, that needs to be regulated. Up to now, there is no formalized process for choosing what can and cannot be used during implementation. However, the builds are done through pipelines, and the used libraries are checked there. There are also weekly tests for library vulnerabilities and third-party software, and libraries are updated at least once a year before the release of a big new version of the product. A password manager is used for shared secrets. It enforces the update to a new version, if available.

OWASP Secure Coding Practices

This year, the developers decided to start using the OWASP Secure Coding Practices to ensure the code is safe and there are as few vulnerabilities or security flaws as possible. A senior developer was assigned the task of analyzing the current state and teaching all the other developers what they should change and do better. Only the analysis was done until the time of writing this thesis.

3.5 Testing

The developers themselves write tests to check the functionality of their code before sending it to testers for black-box testing. They write integration tests (running a part of code including the database and

connections), unit tests (running just methods – right after commit and the developer who committed gets notified in case of an error), and API and rest API tests. Integration tests are the rarest ones, as they take a long time to write. All the written tests are also run during the building process, and if the build is successful, the code should be all right. There are also security tests during the building process, and they most usually check for credentials or other secrets in the code, or they check the third-party software used in the code.

As there is no penetration tester in the company, most testers usually check for edge cases, such as using special characters, but not for security flaws in the system. Penetration tests are done once a year before the new big version is released. However, these are done without logging in to the system. Some customers choose to test the application for security flaws themselves. Usually, the developers get the results and have to correct them. Furthermore, regular vulnerability scans with and without knowledge are performed on the product. When a vulnerability is found, the security team discusses it with a developer, and when it is not a false positive, the developers patch it.

3.6 Deployment and Maintenance

Every developer commits their changes to the branch they are working on. To merge it to the main branch, each developer can make the request, but every developer can also confirm it. There is no dedicated person to approve the merge requests and go through the changes. When a new build is run, libraries are checked, and other basic security flaws, such as credentials or secrets in the code, are also checked. The computer that the builds run on has regular updates of the coding environment but nothing else. In the future, however, the builds are to be run differently. Before the latest successful build is chosen as the final version to be released, it has to be tested thoroughly again. This time, not really the new changes are tested, but mainly the main functionalities. This is partly done by automated tests written by the testers and developers. However, part of the "big testing" is still manual based on written test scenarios.

The product versions are released to the company website at times specified in the release schedule. They are uploaded as a group of

files, and the customers can download the files they currently need (of products and modules they are currently using and have licenses to). Since there are always at least two versions of the product being developed at the same time and not every build is the final one for the given version, only the last couple of builds for each version are kept. The released versions are all stored and kept in order to be able to possibly inspect them later on.

3.7 Security Strengths

During the security analysis of the SDLC, not only the weaknesses but also some strengths considering the security became clear. These are described below and can be considered as the strong core of the SDLC, upon which the improvements can be later built.

Security audits and certifications

The overall security of a company can be managed by different certifications, but perhaps the most common one is the adoption of ISO 27001 and, with that, an information security management system (ISMS). Even though it is not a legal requirement, the company conducts regular internal and external audits and renews its security certifications, such as the already mentioned ISO 27001 or SOC 2 Type 2. Application security can also be strengthened by the certifications. ISO 27001, for example, stresses the need for the segregation of duties, password management systems, user management, including privileged access and user provisioning, and the separation of the different environments used during the SDLC [27].

Endpoint management

All company devices are managed centrally by the company's IT team. The IT team gets notified if a company device is non-compliant based on the policies they set up. The users cannot change the configuration, install malware, etc. Every device is also encrypted, and turning off the encryption again notifies the IT team.

Regular code review

A code review is performed for each new change to the product. With that, at least two different developers see the code and are more likely to discover a possible security flaw. Also, new developers performing a code review can learn from the more experienced ones and see how to code better and prevent possible security or other errors, making their future changes more likely secure.

Regular code refactoring

Even though the capacity of the developers is limited, they try to improve the code constantly. Code refactoring belongs to this, as rewriting the code to be more consistent and readable can also lead to discovering previous security flaws and preventing future ones.

Automated tests

The product is mainly tested manually by the testers. However, there are also automated tests. These span from the tests the developers write and run with each new build to the automated tests of main scenarios that the customers need and that are run before the release of a new version. This leads to the testers being able to focus more on their other testing and have more time to even try the product for security flaws.

Regular vulnerability scans

Before every sprint ends, the product is scanned for vulnerabilities with prior knowledge (as a logged-in user). Extra vulnerability tests are also performed monthly on the production environment. These are done without prior knowledge. The results of vulnerability scans are analyzed, and if a new vulnerability is discovered, the developers are notified and have to analyze it further to possibly solve the problem.

Known future risks solved proactively

If a new version of a technology used in the product is presented, the developers usually try to incorporate it proactively into the product.

An example of this proactive approach is switching to the new version of .NET even though the older one is still supported since the old version would present possible security vulnerabilities in the future.

Certificates for released versions

Each released version and the files included to install it are signed with a strong FIPS 140-2 standard certificate, and there are also other certificates from Microsoft used. The customers can be assured that the files are uploaded by the company and are safe to install.

3.8 Main Security Flaws

Some security flaws were already visible during the security analysis of the SDLC. However, when considering secure development, we have to look at the development as a whole – from the tools, programs, and libraries used all the way to the regular security training. In this section, the main flaws that were found will be described, even though they do not necessarily influence a specific stage of development.

Security considered too late in the SDLC

Even though it should be done right in the first stages of the SDLC, security is mostly first considered during the Design or even Development phase. With that, possible security implications can be discovered too late or will just be more expensive to remediate.

Planning and analysis done by non-technical personnel

It was already mentioned in the security analysis of the SDLC that the stages of Planning and Analysis are mainly done by the PM team. With the fact that developers are not necessarily present in these stages, the resulting change and mainly its possible security implications on the rest of the product are not considered beforehand. It is possible that if a developer was a part of this process, they could catch some possible vulnerabilities early on and not first in the later stages, keeping the price of the fix lower.

No threat modeling or data flow diagram

The developers do not do threat modeling and do not keep an updated data flow diagram. With the product getting bigger and more complex, it can become a problem because some possible vulnerabilities can go unnoticed without knowing all the data flows and connections in the system. This can be especially a problem for new developers who are not as familiar with the product as the more senior developers.

Junior developers can have very big responsibilities

Not only can newcomers be responsible for product changes from beginning to end, but they can also do code reviews. Although this can be a good way for them to learn more about how the more skilled developers code, they can overlook a possible security flaw in the code due to their lack of experience. Furthermore, the fact that a junior can do a code review for a junior developer can lead to a problem with an even bigger probability.

No security baseline for development environment

The developers install the development environment themselves and can decide which extensions to use (only the ones the company can use license-way, of course). There is a set of recommended extensions; however, it is not enforced, and the suggestions are not really security-oriented – they are more about the cleanliness of the code, etc.

Update of development environment up to developer

On top of the environment security baseline not being formalized, its update is not enforced. Every developer updates their respective environment whenever they want. That means that some extensions or even the integrated development environment (IDE) itself can be missing some security updates. However, it is commonly understood between the developers that they should update as soon as a new update is available.

Temporary developers use their own laptops that are not company-managed

The company has a security policy that also discusses the Bring Your Own Device (BYOD) policies; however, it does not specifically focus on the development environment of these devices. Another problem is that the devices are not managed by the company, so the security baseline that is otherwise enforced on the devices does not apply to them.

Anyone can approve merge requests

No dedicated person oversees the merge request approval, meaning that no developer goes through everything that is put into the product. That should not be such a big problem, as every change goes through code review. However, the fact that the requester themself can approve the request can definitely lead to back doors, etc.

No formalized process for choosing third-party software

Similar to the extensions for the development environment, there is no formalized process that the developers have to follow when choosing a third-party software to use in the product. That means that no security analysis is done beforehand. However, if the software to be used has known vulnerabilities, the developers look for alternatives until these are remediated. Open-source software can also be used. However, this does not necessarily pose a threat and can be even more secure than if the developers developed everything in-house.

Non-existing software bill of materials (SBOM)

In the earlier versions of the product, the used libraries were documented. However, with the increased complexity of the product, this list is not updated anymore, and there is also no comprehensive SBOM describing all the different open-source and third-party components. Supply chain attacks are more likely if the SBOM is not up-to-date, as some problems can be overlooked or some components can be forgotten about.

Not every vulnerable library is replaced

When a new vulnerability is found in one of the libraries or other third-party software used in the product, it should be replaced or updated immediately when a security patch is released. However, due to the complexity of the product and the dependencies, this cannot always be easily done. That said, if the vulnerability is critical, it is a priority to deal with the replacement or update.

Code review is not always performed after returned for correction

As already mentioned in the analysis of the SDLC, when a tester returns a change for correction, it then goes back to the testing stage once the error is fixed. Even though a code review is encouraged before sending the change to testers again, it is not mandatory and sometimes is not performed due to different reasons.

No software composition analysis (SCA) or static application security testing (SAST) used

Similarly to the missing SBOM, SCA focusing on analyzing and managing the open-source components within a software application is not used. Even though a DAST is regularly used for the vulnerability scan of the running application, an SAST that analyzes the code itself is not used.

No penetration testers

The testers team is relatively small, and there is no specialized penetration tester. The penetration tests are performed regularly by an external company; however, they are only tests without prior knowledge. Customers sometimes also perform penetration tests on their environments; however, this may be too late, and the company should not rely on these.

No regular or mandatory security training for developers

The developers have to undergo the mandatory regular security training delivered by the security manager to the whole company. This

training does not specifically concern the secure development or secure coding principles. The developers have the opportunity to learn themselves, with four hours per sprint allocated for self-improvement. However, these hours are usually spent on new technologies or even skipped, as not every developer has, in the end, enough time outside their regular work. New secure coding principles or possible vulnerabilities can also be discussed in a regular meeting dedicated to interesting topics that could interest the whole team, but that is mostly not the case. Other than that, there is no mandatory security training.

Security patches may take a long time

The capacity of the development team for unexpected changes is limited due to the low number of developers. When a new vulnerability in the product is discovered, it can take a relatively long time for the developers to fix it. Other than that, there is only one developer who is really knowledgeable about secure coding and different possible vulnerabilities and attacks and how to prevent them. That means that security is mostly consulted with this developer, and when they do not have time, the process may slow down even more.

4 Choosing the Best Framework

When choosing the most suitable framework, its usability has to be considered first. Focus points and metrics must be established to compare the presented methodologies and frameworks. Furthermore, the chosen framework should cover as many of the security weaknesses found as possible and be able to guide the company in remediating them. At the same time, it should be able to build upon the found security strengths. The frameworks are compared based on these requirements in the following sections.

4.1 Focus Points and Metrics in General

First, the criteria were chosen after a discussion with the security manager, who pointed out possible company hindrances and gave insight into what they consider when adopting any new tool or process. These criteria are chosen specifically based on those needs. There are several categories of possible metrics – security engineering, business, security characteristics or software system [28] – and all of these were consulted with the security manager.

Secondly, more generic criteria are also considered. Not every framework offers the same number of security processes, features and advice. Therefore, different important security criteria were considered during the selection process.

The main things that have to be considered during the selection process are the level of detail provided in the framework, examples of usage of different processes or practices, and the fact that there are some recommended tools. Since the SDLC is given in the company, the fact that the framework can be adapted to this SDLC was also considered. However, many processes and techniques mentioned in the selected frameworks can be universally adapted to any SDLC, so that was not the main focus.

Another thing to consider was the structure of the framework itself. Is the process described in consecutive steps, and is the order logical for the company? Since the company has limited resources to implement new security measures, including the framework for SSDLC, the number of human resources needed was also a criterion for

selection, mainly if it is possible to incorporate the processes without the need for external security teams.

The parts that should not be absent in the framework are setting up security objectives and requirements, performing a risk analysis, threat modeling, various validation processes (for example, vulnerability and penetration testing) and also the need for trained personnel to oversee any changes and improvements during the SSDLC.

As the human factor is still the most vulnerable part of security, the inclusion of training of the developers throughout the stages of the SDLC (or other involved personnel) is a big criterion for choosing the framework. As one of the best practices, security should be a shared responsibility of all involved personnel [29].

In table 4.1, there is a comparison of the chosen frameworks based on these metrics.

Although Securosis specifically has a chapter on Security Tools for DevOps, these are merely examples of where to test and how to build a toolchain. No specific tools are mentioned there; therefore, the focus point for recommended tools is marked as not met. CSDL recommends some tools, but the Cisco company exclusively develops these, so they are limited. The only framework with recommended tools that are not only made by the company is, therefore, MSDL.

All the frameworks are considered suited for the company because the different practices can be fitted to the stages of SDLC that are already in place. They may not be described as agile; however, they could be fitted to that. Even if all the frameworks do not describe the practices in the phases as are typically described for the SDLC, each framework describes them in consecutive steps. These vary from SSDF's four groups to the MSDL's ten practices, spanning from establishing security requirements to the need for security training. The frameworks that really describe the practices in the phases of the typical SDLC are Securosis and CSDL.

From this comparison, the best possibilities seem to be SSDF, MSDL and CSDL.

Table 4.1: Comparison of chosen methodologies based on the focus points

Methodology / Focus Point	NIST SSDF	MSDL	Securosis	JNSDL	CSDL
Examples of Usage	X	X	X	X	X
Recommended Tools	-	X	-	-	X
Suited for the Company's SDLC	X	X	X	X	X
Consecutive Phases	X	X	X	X	X
Need for External Teams	-	-	-	-	-
Security Objectives and Requirements	X	X	X	-	-
Risk Analysis	X	X	X	-	X
Threat Modeling	X	X	X	X	X
Security Testing	X	X	X	X	X
Security Training	X	X	X	X	X

4.2 Comparison Based on the Found Security Flaws

Other than the general metrics and focus points based on the company's processes of choosing new tools, the framework's inclusion of the security flaws that were found during the security analysis of the SDLC should be considered. In table 4.2, the comparison of the chosen frameworks with regard to anything that would address the flaws is shown. The flaws that are already addressed by the comparison in section 4.1 are left out of this table.

The two flaws, first, that the planning and analysis are mainly done by the PM team with limited technical knowledge, and second,

Table 4.2: Comparison of chosen methodologies based on the security flaws

Methodology / Security Flaw	NIST SSDF	MSDL	Securosis	JNSDL	CSDL
Security late in SDLC	X	X	X	X	X
Planning and Analysis by PM	X	X	-	-	-
Big responsibilities for Juniors	X	X	-	-	-
No security baseline for environment	X	X	X	-	X
No enforced update of environment	X	X	X	-	-
Missing BYOD policy	-	-	-	-	-
Merge request approval by anyone	X	X	-	-	-
No process for third-party SW	X	X	X	-	X
No SBOM	X	X	X	-	X
Lingering insecure libraries	X	X	X	-	X
Code review not always performed	X	X	X	X	-
No penetration testers	X	X	X	X	X
Low capacity for security patches	X	X	X	X	X

that junior developers can have big responsibilities are considered as addressed if the framework specifies the security roles at all. With that, only the qualified personnel with training according to their role should be able to do the jobs during the SSDLC. Securosis describes the role of security personnel during the SDLC, however, it does not describe the roles of the other people and so the flaws are considered as not addressed.

The problem with BYOD policy or overall non-company-managed devices is not addressed by any of the frameworks.

The merge requests are not really the main flaw here. However, there should be a process of securely incorporating all code into the production branch. If the framework somehow describes and controls this process, the flaw is considered as addressed. What is interesting is that Securosis goes in a completely different direction and promotes writing the code straight to the main branch, as implied by DevOps CI.

As for the process of third-party software, Securosis stresses the importance of scanning even these components for vulnerabilities. However, that is the extent of it, and it can be viewed as insufficient compared to SSDF, MSDL or CSDL.

Securosis does not really describe preparing and keeping an updated SBOM. However, the need to monitor and control versions and updates for all parts of the product is mentioned in the part of the white paper about building a toolchain, so that flaw is somewhat addressed.

For the last three flaws, the company's processes have to change, or new personnel need to be hired in order to remove them fully. The flaw is considered as addressed if the need for the root of the flaw (code review, penetration testing, security patches) is described in the given framework.

Based on these results, the best frameworks to address the found security flaws are SSDF and MSDL. Securosis is the third option. However, some of the flaws were addressed only marginally, and the practices would probably not suffice as solutions. Therefore, Securosis is not further used.

4.3 Chosen Framework with Reasoning

Given that some of the described frameworks (or methodologies) are very generic and vague, the choice was from the beginning mostly between the NIST SSDF, Microsoft SDL and Securosis: Putting Security Into DevOps, as these three offer the most specific suggestions for improvement of the SDLC. The further argument in favour of the MSDL was also the fact that the company is developing Microsoft-centred software and using Microsoft throughout the company.

As can be seen in the tables 4.1 and 4.2, the best frameworks based on the comparisons of general focus points and also the found security flaws are the SSDF and MSDL. The two frameworks each describe some of the focus points or solutions to the security flaws in more or less detail. What is very well addressed in MSDL can be more vague in the SSDF and vice versa. With that in mind and the fact that even the two best frameworks based on the comparison do not seem perfect for the needs of the company, the mix between them – NIST SSDF and MSDL – was chosen.

As this is the first iteration of implementing specific secure processes in the SDLC, it is not expected to be perfect. Not all the practices described in the frameworks and methodologies can be implemented at once and can be added in later iterations.

5 Implementing the NIST SSDF and MSDL

In this chapter, the implementation of the NIST SSDF together with MSDL is described.

The implementation is for the sake of this diploma thesis done with regard to the current time, personnel, and money capacity of the company. The recommendations in this thesis were consulted with the security manager but will need to be discussed with more people, especially from the development team, before actually being incorporated into the company.

In order to be most effective, the recommendations are written up into a policy for secure development – the Secure Software Development Lifecycle Policy. With that, every developer or person involved in the SDLC can easily find what to do and what to look out for security-wise during each stage of the SDLC. There is also an extra policy specifically for the developers concerning the secure environment setup and secure coding principles, called the Secure Coding Policy.

In each policy, the objectives, scope, roles, and responsibilities are written before the content based on the frameworks is further described. These are for the policy as a whole, not for the frameworks themselves.

These policies can be found in the electronic attachments to this thesis, and the recommendations are also written out in the following sections.

5.1 Preparation for the Implementation

Before the SDLC itself can be improved, there are some steps that need to be made that do not align with any of the SDLC phases. All the steps described should be taken into consideration before planning each new big version of the product. As a part of the preparation, the policies should also be reviewed and updated at least annually.

For this phase, the SSDF practices PO.1.1, PO.1.2 and PO.2.1-PO.2.3 and MSDL practices 1, 4 and 10 are implemented in the new policy.

Define Security Requirements for Software Development (PO.1) is altogether mostly implemented by the newly written policies that will be shared in the company for all personnel responsible for activi-

ties during the SDLC to read and adhere to. Namely **PO.1.1** and **PO.1.2** stress the importance of security requirements and policies. Of course, the two policies do not suffice for the whole management of SDLC, so throughout the policies, many other internal policies and processes are referenced, such as vulnerability management, risk analysis, backup policy, threat modeling, etc.

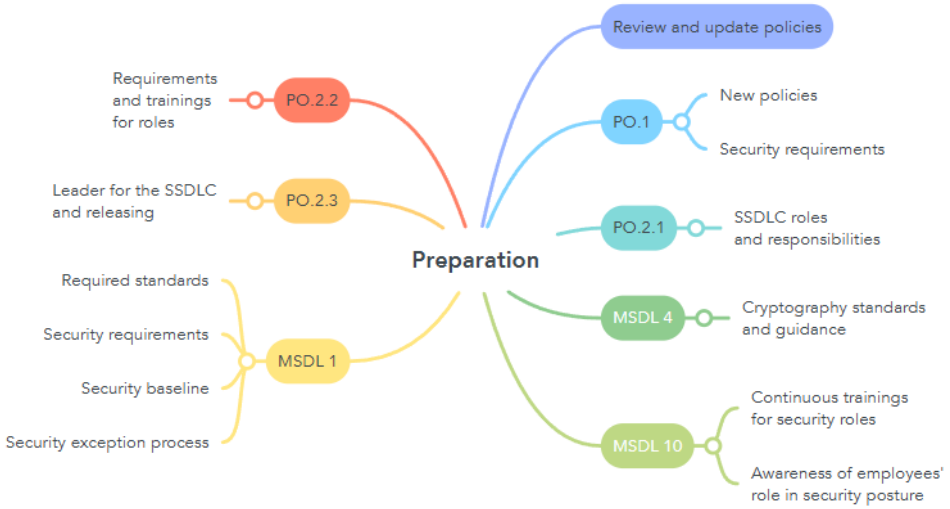


Figure 5.1: Practices for the phase Preparation

When implementing the **Implement Roles and Responsibilities** (**PO.2**), new roles are defined based on **PO.2.1**. These are not the same as mentioned previously in the policy for the whole company, but they are specific roles for the SSDLC. The five main roles are Code owner of Service Desk (main developer of SD), Code owner of Asset Management (main developer of AM), Product Owner (product manager), Senior management (manager of development) and Security champion (developer with highest security knowledge). The roles and their responsibilities have to be checked and updated annually in a scheduled ticket. The Security Team conducts the review and verifies that the roles actually do their responsibilities.

For **PO.2.2**, the requirements for each role are specified, and for them to be able to fulfil these, regular training needs to take place. It is not specified how exactly the training is to be conducted, so it can be internal or external training. Some of the requirements are

already fulfilled by the roles. However, continuous training leads to a deepening of the security knowledge, and that is extremely important for responsible personnel.

PO.2.3 calls for a leader for the entire SSDLC, as well as releasing. In the company, these two roles are not on one but two people. The person responsible for the whole release is to be the release manager and the manager of development is responsible for overseeing the whole SSDLC.

The first practice of the MSDL is called **Establish security standards, metrics, and governance**. This practice overlaps with SSDF's PO.1 in some points. However, there are also new recommendations. The two new policies, as well as the chosen frameworks themselves, ensure that the company has recognized the required standards (**1.1**) and defined security requirements (**1.2**). The baselines are the NIST SSDF, MSDL, and OWASP Top 10[1]. Security Exception Process (**1.4**) is handled by several steps in the company. Short-term non-compliance with the internal policies is handled by informing the Security Team with an explanation, short-term actions to reduce potential risks, and a remediation plan to achieve compliance again. The remediation of important and critical vulnerabilities is to be done until the next sprint. Furthermore, if a risk cannot be remediated at the time, the security champion, senior management, and code owner of that vulnerability need to agree to accept the risk.

In the fourth MSDL practice called **Define and use cryptography standards**, the importance of cryptography standards and guidance is stressed. The company already has an internal Policy of Cryptography Standards that is based mostly on the National Cyber and Information Security Agency (NÚKIB)[2] and NIST recommendations and best practices. Therefore, only the need to secure the data in chosen data centres and transport technologies is added to the new policies.

Lastly, the tenth MSDL practice **Provide security training** is incorporated into this phase of the SSDLC. This one interlaces with **PO.2.2**. The chosen security roles should now be trained at least annually specifically for their role – meaning on top of the security training mandatory for all employees of the company. On the other hand, not

1. `https://owasp.org/www-project-top-ten/`
2. `https://nukib.gov.cz/`

only do the employees need to be aware of their security posture, but everyone is to be aware of their role in the security of the product and is encouraged to further learn by themselves.

5.2 Improved SDLC

5.2.1 Planning

For the Planning phase, parts of the NIST SSDF PO.4 and MSDL first practice can be implemented. The practice **Define and Use Criteria for Software Security Checks (PO.4)** together with the first MSDL practice **Establish security standards, metrics, and governance** are implemented by requiring the definition and documentation of Key Performance Indicators (KPIs) and security requirements during this phase for sprints and also new versions of the product. These later serve as a baseline to be able to determine if the developed product is secure and the SSDLC was successful or if an improvement plan for future development needs to be presented. The security criteria are also to be documented in the DoD for all developers to consider before proclaiming a new change as finished.

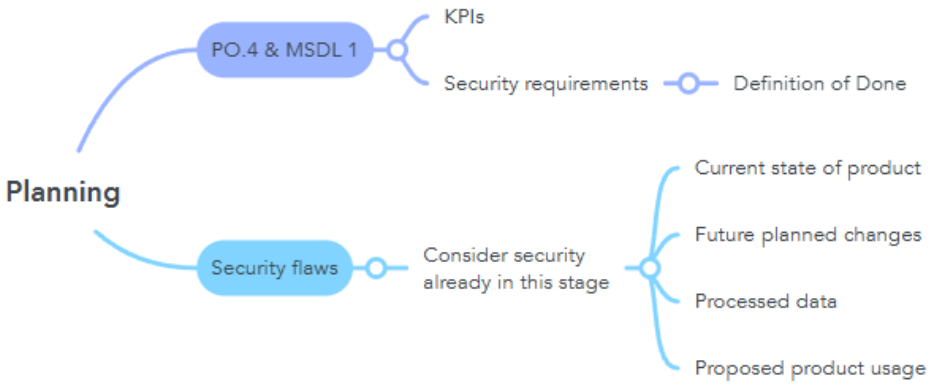


Figure 5.2: Practices for the phase Planning

Other than that, security flaws are also taken into consideration. The biggest change is that security should be considered already during this phase, as this was not the case before. The current state of the

product, as well as the future planned changes, need to be considered from the security point of view. Circumstances such as the processed data or proposed usage of the product need to be considered before planning a new change in order to evaluate the security implications sufficiently.

5.2.2 Analysis

For this phase, no specific practices from NIST SSDF or MSDL are implemented. Even so, the phase needed improvement, so some recommendations were put in place in the policy. The main recommendation is that if the PM team (who mainly does the analysis) is unable to analyse the security implications of the proposed change, they need to consult one of the security roles that should have broader security knowledge.

The risk analysis of a change is to be done during this phase as well. Risk analysis should be done from the viewpoint of the whole product and how the change will influence it. If there is a new risk to the system found during the analysis, its assessment needs to be done — either remediation or well-reasoned acceptance needs to be noted as an outcome and approved by the senior management and security champion.

The threat model and architecture overview (or at least the data flow diagram) need to be consulted during the analysis of each change, and if the models need to be updated or changed, that is to be added to the overall analysis with instructions as to how. With that, the developers can keep the models up-to-date in the later phases.

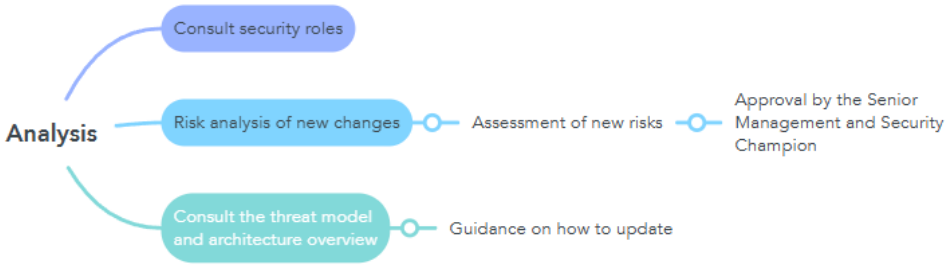


Figure 5.3: Practices for the phase Analysis

5.2.3 Designing

For the designing phase of the SSDLC, the results of the analysis need to be clearly given to the developers so they can correctly incorporate them into the design. For this phase, the SSDF PW.1.1, PW.1.2 and PW.2 and MSDL third practice have been implemented.

According to **Design Software to Meet Security Requirements and Mitigate Security Risks** (**PW.1**) and specifically **PW.1.1**, the threat model needs to be updated with each change if that is the result of the analysis. The threat modeling is to be done based on the OWASP Threat Modeling Process [3] with the help of Microsoft Threat Modeling [4]. All the previous risks and vulnerabilities should also be taken into account when designing a change. For **PW.1.2**, in case of new risk, the design decisions, along with risk responses and possible exceptions from compliance with policies, need to be well documented. All exceptions should also be regularly re-evaluated.

Review the Software Design to Verify Compliance with Security Requirements and Risk Information (**PW.2**) is self-explanatory. The CAB needs to review and approve the design and updated threat model. All flaws found during the reviews must be corrected if they lead to failure to meet the requirements. If the design cannot meet the security requirements, it should be changed altogether, as the requirements are non-negotiable. If there are any residual vulnerabilities, their mitigation needs to be documented together with the other findings for possible audit trail.

MSDL third practice **Perform secure design review and threat modeling** is similar to the **PW.1.1** but also counts for the case when the whole threat model cannot be changed at the moment due to its difficulty. In that case, at least the Data Flow Diagram must be updated. All architecture analysis should be done from the attacker's perspective, and the found threats should be rated and prioritized in order to determine what is needed to improve next. The existing mitigations should be described during this phase, and if there are no mitigations yet, they need to be created as tasks for further development.

---

3. `https://owasp.org/www-community/Threat_Modeling_Process`
4. `https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling?oneroute=true`

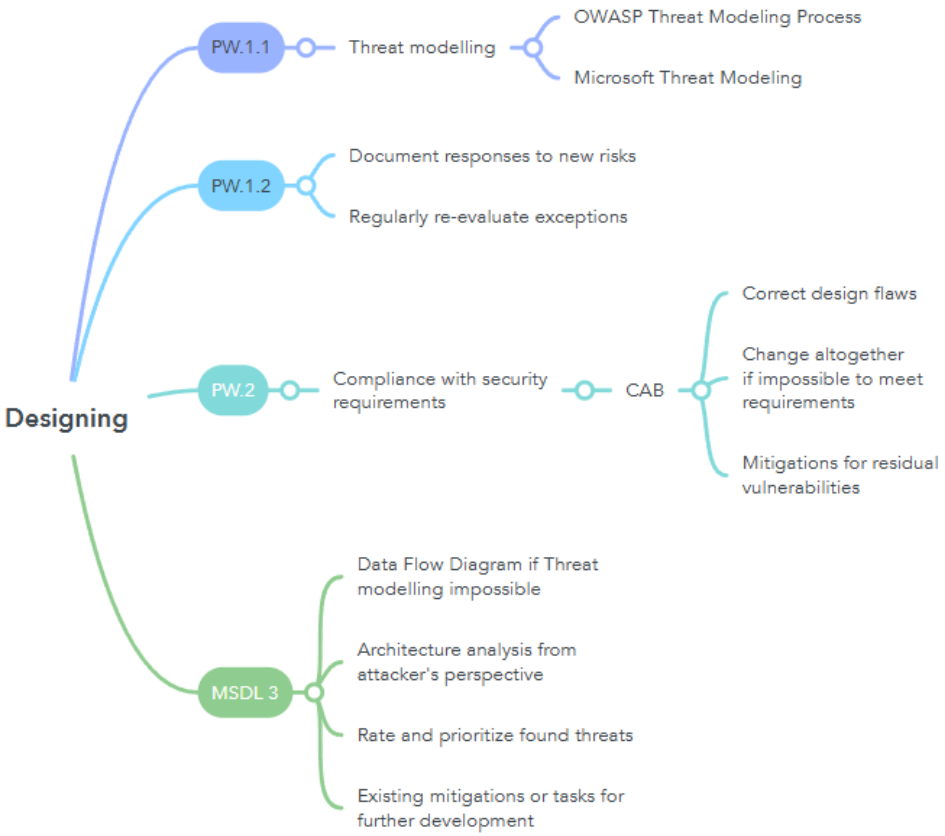


Figure 5.4: Practices for the phase Designing

5.2.4 Implementation

This phase, together with the phase of testing, is described in the Secure Coding Policy and not the main SSDLC Policy.

Environment

For the development environment, the SSDF practices PO.1.3, PO.3, PO.5, PW.6 and MSDL second and sixth practices are implemented. The last part of the **Define Security Requirements for Software Development** (**PO.1**) practice is focused on requirements for third-party software. The security of each third-party component needs to be analyzed by searching for previous vulnerabilities, data leaks caused by

the component, etc. If there is a record of such occasions, the response to that needs to be considered in order to decide if the component can be used — namely, the time to patch the vulnerability, the publisher's transparency, etc. If the response is not up to the company-defined standards, the component cannot be used in the system. This also applies to open-source software.

Implement Supporting Toolchains (PO.3) is heavily supported by the **PO.1.3** practice, and so every tool used in any toolchain has to be chosen based on the same principle as third-party components. The tools also need to be regularly updated, upgraded or replaced if a vulnerability is found. The tools' logs need to be monitored, and an audit trail should be available for each one.

The practice **Implement and Maintain Secure Environments for Software Development (PO.5)** sets the baselines for a secure environment. All the different environments used during this phase need to be separated, and access to these environments needs to be managed and minimized. On top of privileged access logging, multi-factor authentication (MFA) and network segmentation need to be used when possible. Monitoring for attacks (be they only attempted or real ones) should also be done continuously. Furthermore, the devices used for anything related to development should be company-managed by the IT team, which also sets a security baseline for endpoint management. In case anyone needs to use their own device, the internal Bring Your Own Device (BYOD) policy gives guidance on the correct usage. This should, however, be limited.

Configure the Compilation, Interpreter, and Build Processes to Improve Executable Security (PW.6) is again based on the **PO.1.3** practice, and so all compiler, interpreter, and build tools should be used according to that practice. Furthermore, they again need to be updated to the newest stable version as soon as possible. The compiler should be chosen so that it can warn the developer of unsecured code during compilation. The building process should end with zero errors and possibly zero warnings as well. Each such remaining warning should be analyzed and possibly eliminated. The building environment has to be secured, with limited access and updated regularly. If possible, the overall configuration should be regularly tested and approved.

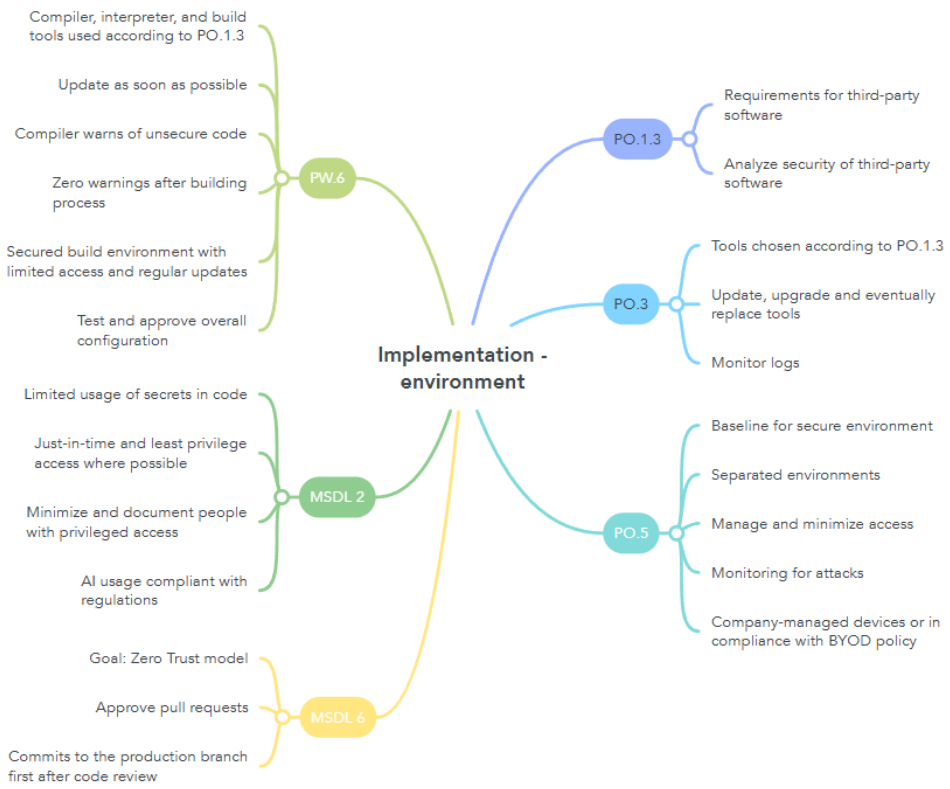


Figure 5.5: Practices for the phase Implementation – Environment

The second MSDL practice **Require use of proven security features, languages, & frameworks** focuses on the solutions also based on Microsoft products – for example, the usage of Managed Identities of SAS tokens should be considered. In the code, the usage of secrets should be limited, but when necessary, it must never be in plain text. Secure Credential Storage should also be used. On top of managed access, Just-in-time and least privilege access should be used where possible. The number of people with privileged access must be minimized and documented. With the growing popularity, the security of AI usage should be considered even in the SSDLC. When AI is being used in the product, be it for development or in the product itself, the national and international legislation, as well as the internal security rules written in the internal Security Policy, apply.

Lastly, the sixth MSDL practice **Secure the engineering environment** is implemented in this part. The goal for the developers should be to strive for a zero-trust model, meaning that each access request should be treated as if it came from an untrusted network. Pull requests should be approved, and commits to the production branch should not be possible without code review.

Implementation

The implementation itself is improved by the SSDF PS.1, PW.4 and PW.5 practices together with the fifth MSDL practice.

The practice **Protect All Forms of Code from Unauthorized Access and Tampering** (**PS.1**) focuses on the storage of the code itself. The code is to be stored only in a repository with version control and other storage, especially the devices of the developers or any removable media is forbidden. Access to the code also needs to be restricted, and only permitted personnel, mainly the developers or quality assurance (QA) personnel, should be permitted to access it. The person who made a change to the code is responsible for it, and all the changes are to be documented, with the possibility of tracking down who made what change. The person with the code owner role should have an overview of all the changes to the product and approve the merge requests. They do not need to approve every single change, as the code review process is used for that.

The next practice, **Reuse Existing, Well-Secured Software When Feasible Instead of Duplicating Functionality** (**PW.4**), should lower the costs and also decrease the possibility of new vulnerabilities in the code by not creating new components if other established well-secured components can be used. These third-party components need again to be chosen based on **PO.1.3** practice. When it is really impossible to use such components, well-secured software components that comply with the SSDLC processes should be created in-house. On top of that, a list of forbidden third-party software components needs to be created, updated regularly, and stored in a place where all developers can access it. The list can be based on found Common Vulnerabilities and Exposures (CVEs) or regulations from state organizations such as the NÚKIB. All of this contributes to the complete Software Bill of Materials (SBOM), which should also be kept up-to-date and available

to every developer. The details about the supplier, the current version of the component in the product, licensing, dependencies and security status with known flaws and their fix must be included with every component in the SBOM.

For the practice **Create Source Code by Adhering to Secure Coding Practices (PW.5)**, the OWASP Secure Coding Principles serve as a baseline. Another way of securing the code is checking for the known vulnerabilities of the used development languages and environment and keeping them in mind when developing the change. Each developer should also read their code again and correct security flaws according to the DoD before sending it for code review.

MSDL practice **Secure the software supply chain** focuses on open-source software (OSS). If a dependency on any OSS exists in the code, it should be inventoried together with possible vulnerabilities. The OSS needs to be included in the SBOM.

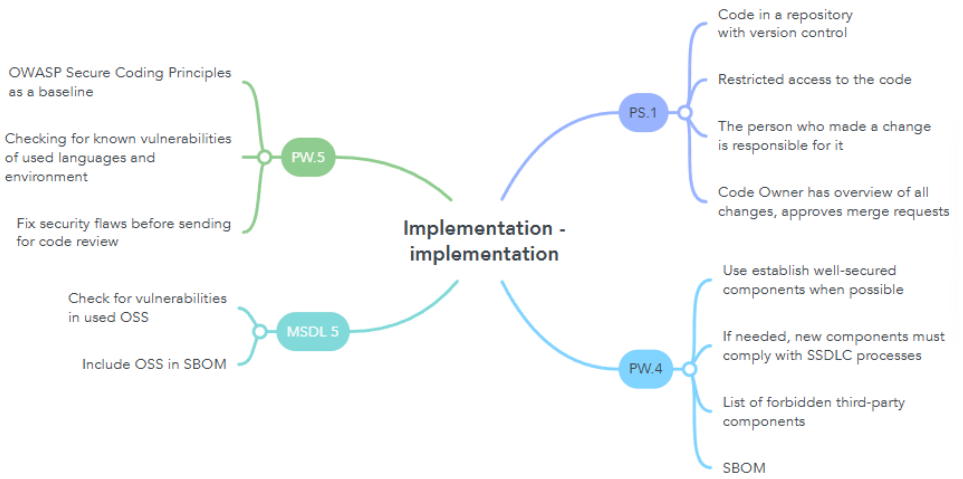


Figure 5.6: Practices for the phase Implementation – Implementation

Code review

Code Review is newly managed by the implementation of SSDF PW.7 and MSDL second practices. For the **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**, the rules of the code review are implemented. The code review is to be done after implementation of

every new change and also when the change is returned for correction by a developer who was not involved in the design and implementation of the change. Junior developers should not perform code reviews for other junior developers; however, if that is needed or desirable, another senior developer must perform an additional code review. The feedback from the code review is to be marked visibly in the code or in the ticket for the change. All feedback should be incorporated into the correction. The DoD and the Secure Coding Policy give guidance on how to perform the code review. On top of the code review itself, each new change should also be scanned for issues.

The practice **Require use of proven security features, languages, & frameworks**, previously also implemented in the defining of a secure environment, is here implemented with regard only to the code review. The developer conducting the code review needs to check the use of proven security features, languages, & frameworks according to **PW.6** practice. If the reviewer finds non-compliance, that is also a reason to return for correction, other than finding programming mistakes.

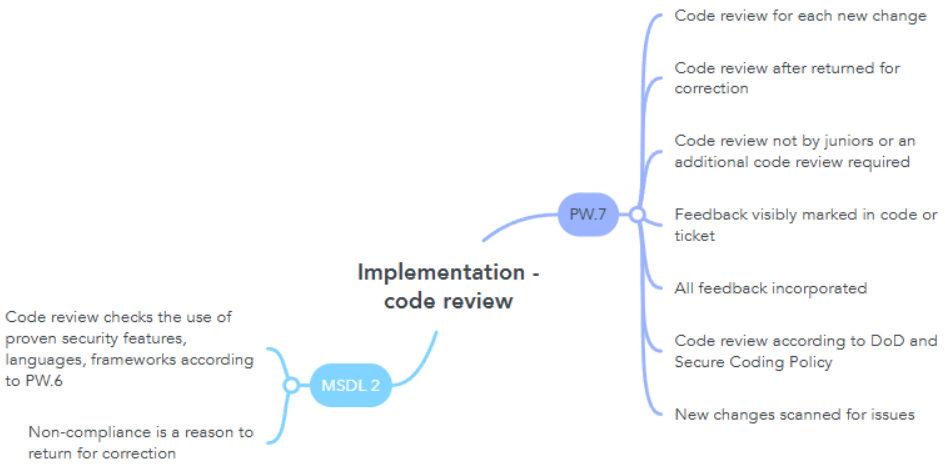


Figure 5.7: Practices for the phase Implementation – Code review

5.2.5 Testing

As there are already tests done during the SDLC, the practices in this phase, namely NIST PW.8, RV.1.2 and RV.1.3 and MSDL seventh prac-

tice, mainly add to these tests and formalize them. Generally speaking, the unit and integration tests should be run with each change. The configuration, especially the operating environments, should also be tested before the new version is sent to production. The big versions of the product must also go through overall testing of at least the main functionalities right before being published. This should be done on top of testing each change as it is introduced. The main thing is that the QA team must test the changes with security implications (which should be newly described in the design of the change) in mind and test not only the functionality but also the security of the change.

The practice **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)** is not strictly required. However, the executable code testing should be considered and used where possible. It can be used as the last step to find any remaining vulnerabilities, and the results, together with root causes, should be documented in order to limit them in the future.

The practices RV.1.2 and RV.1.3 from **Identify and Confirm Vulnerabilities on an Ongoing Basis (RV.1)** are implemented by the need for automated code analysis with SAST tools to be performed regularly before each sprint ends. The SCA should also be newly performed. The whole process of vulnerability management is further described in the internal vulnerability management policy, and all vulnerabilities found need to be accessed according to the policy. The specific tools are not a part of the policy, as that would lead to the constant need for new versions of the policy in case the tools change.

Although DAST testing is already performed in the company, the MSDL seventh practice **Perform security testing**, which mentions many test types to consider, also mentions it, and so now the DAST testing will be formalized in the policy. The security team is responsible for that testing, which contrasts with the other tests performed by the development department (including testers). The found vulnerabilities again need to be assessed according to the vulnerability management policy. Another test type mentioned in the practice is penetration testing. It needs to be performed regularly, which is to be performed annually by a certified external company. For future development, internal penetration testing should also be considered, and new personnel be hired, or current employees should be trained to perform it.

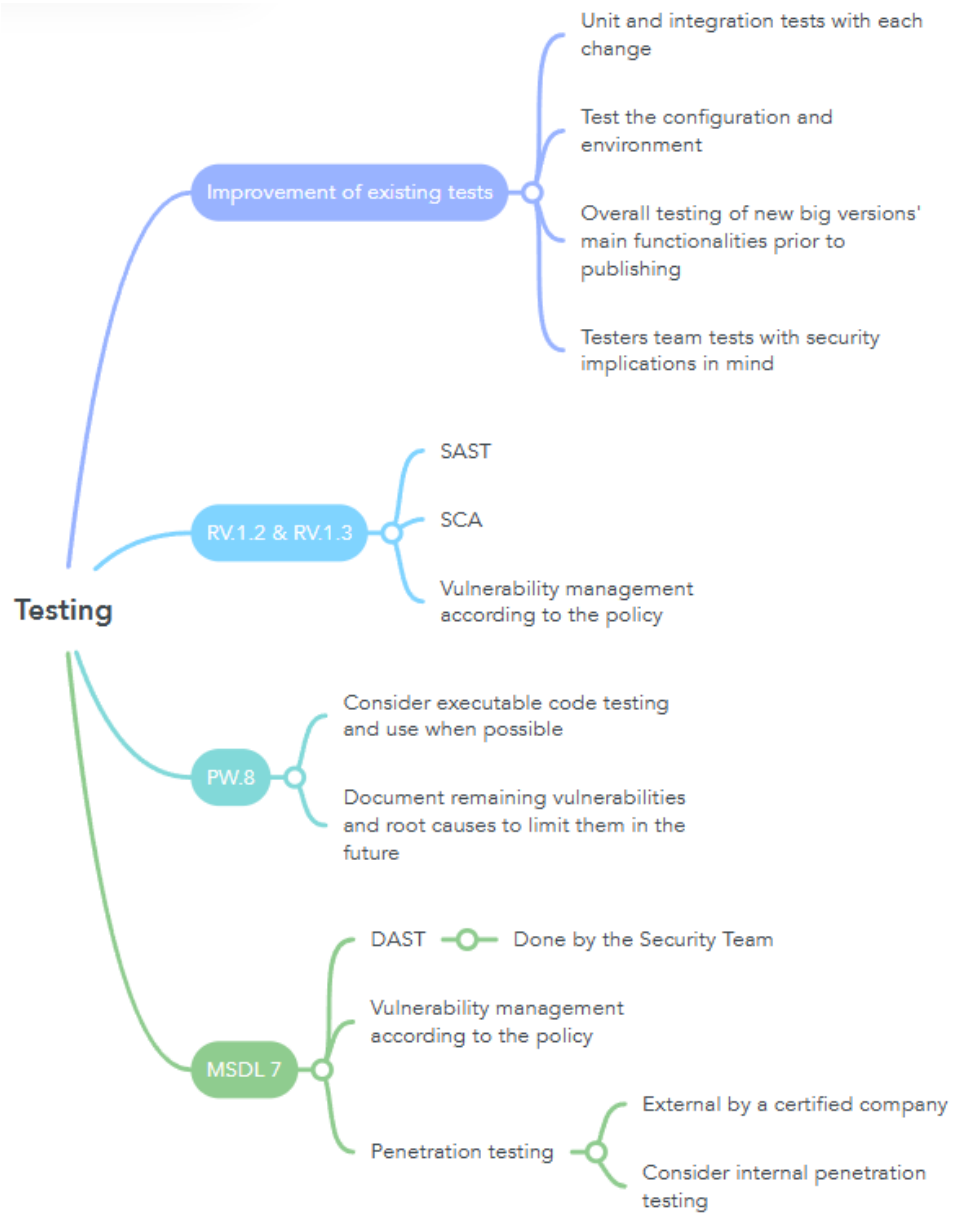


Figure 5.8: Practices for the phase Testing

5.2.6 Deployment

This phase is no longer managed by the development team only but also by the operations and security teams. To help with the management, the SSDF practices PO.4, PS.2, PS.3 and MSDL eight practice are implemented.

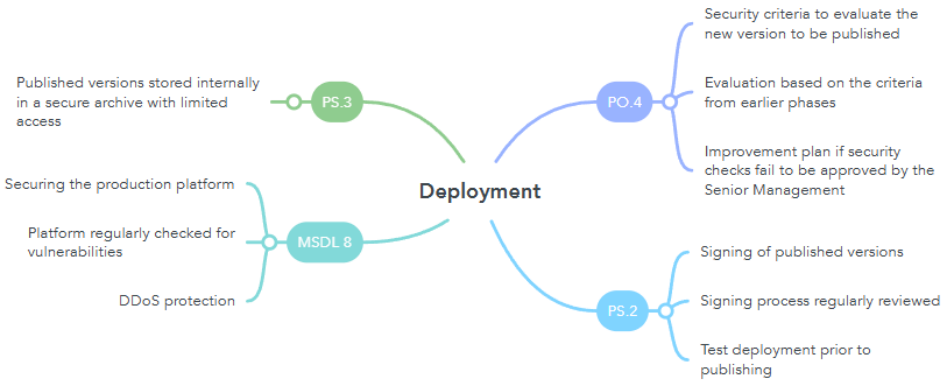


Figure 5.9: Practices for the phase Deployment

The practice **Define and Use Criteria for Software Security Checks** (**PO.4**) ensures that security criteria are defined to be able to evaluate the new version to be published. It is to be evaluated based on the number of vulnerabilities, new risks to the system and their severity. It is also to be evaluated based on the requirements set at the beginning of the SSDLC. The evaluation should be documented, and if the security check shows the system security does not meet the requirements and KPIs set at the beginning of the SDLC, the security check is marked as failed. In that case, an improvement plan for further development needs to be presented and approved by the senior management.

To **Provide a Mechanism for Verifying Software Release Integrity** (**PS.2**), each published version needs to be signed. With that, the customers can verify that they downloaded what they were expecting. The signing process needs to be regularly reviewed in order to ensure the authenticity and validity of the certificate. Furthermore, a test deployment is to be done prior to publishing the version to the customers. The release manager manages this according to the release schedule.

The certificates should also be checked during the test deployment to ensure their correctness and validity.

For the practice **Archive and Protect Each Software Release (PS.3)**, each published version is also to be stored internally in a secure archive with limited access. Only the release manager should be able to access the files. They can approve access to other people if needed.

MSDL practice **Ensure operational platform security** is implemented by securing the production platform with MFA and role-based access control (RBAC). On top of that, Just in Time privilege elevation (JIT) should be used where possible, and the platform itself should be regularly checked for vulnerabilities. Lastly, DDoS protection should also be enabled.

5.2.7 Remediation and Maintenance

For the last phase of the SSDLC, the NIST practices RV.1, RV.2 and RV.3, together with MSDL ninth practice, are implemented. The Security and Operations teams mainly oversee this phase.

The teams need to **Identify and Confirm Vulnerabilities on an Ongoing Basis (RV.1)**. For that, all the software documented in the SBOM needs to be monitored for new vulnerabilities, and in case a new vulnerability that could compromise the customers' data is found, the customers need to be immediately informed. The way of informing the customers varies based on the model of their license. For the customers using the product with SaaS licenses, the SaaS team serves here as the Product Security Incident Response Team (PSIRT) and needs to publish the vulnerability together with the planned date of remediation on the SaaS customer portal. For the on-prem customers, an immediate message needs to be sent to them again with the planned date of remediation. The internal incident response plan determines the response and is regularly tested and updated based on the test results by the security team.

To be able to **Assess, Prioritize, and Remediate Vulnerabilities (RV.2)**, each newly found vulnerability has to be documented by a ticket in the Service Desk. Next, risk calculations and possible remediation, together with the solution itself, are also monitored in the ticket. With that, everything is in one place, and it is easy to find and look through vulnerabilities to see what has the highest priority. If it

is decided that the risk will be accepted or transferred, the senior management needs to approve it, and the approval is to be documented in the ticket as well. A plan for future fixes needs to be prepared for each risk that cannot be remediated at the moment. The stakeholders need to be informed of such known vulnerabilities that stay in the product.

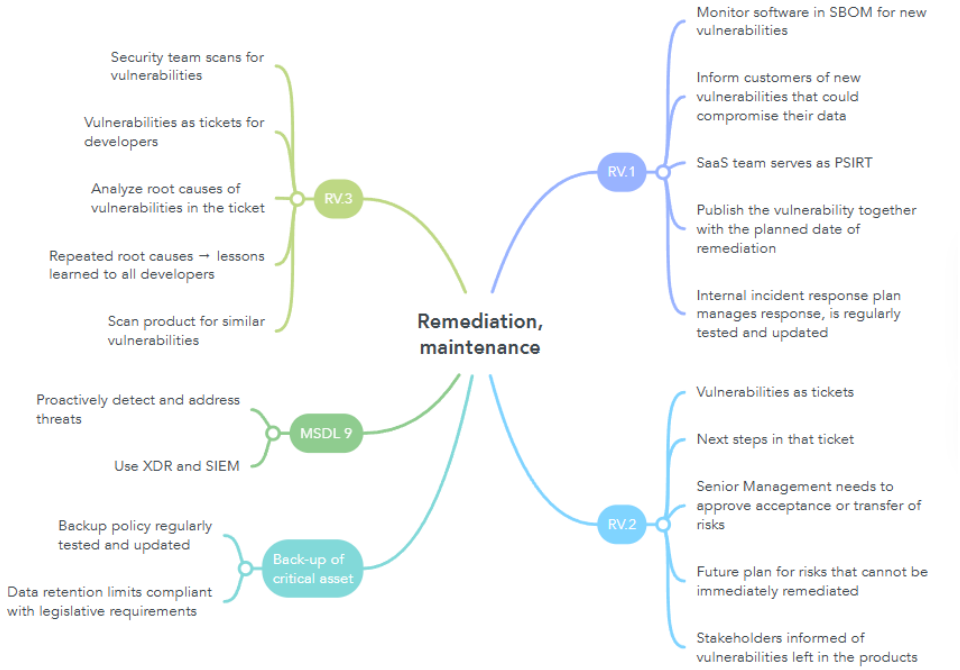


Figure 5.10: Practices for the phase Remediation and Maintenance

The practice **Analyze Vulnerabilities to Identify Their Root Causes (RV.3)** is implemented so that the security team, in accordance with the internal vulnerability management process, scans for and gathers information about new vulnerabilities in the product. This should include all software in the SBOM, including third-party software. For the discovered vulnerabilities, the developers need to analyze their root causes and again record them in the ticket of the vulnerability. For repeated root causes or root causes that have the potential of being repeated, the developer needs to sum up the lessons learned in a document available to the whole development team or present it at the meeting dedicated to knowledge sharing between the developers. However, that is not always enough, and possible mechanisms to pre-

vent the next instance of the root cause need to be implemented, and the product should be scanned for similar vulnerabilities immediately.

The practice **Implement security monitoring and response** stresses the importance of proactively detecting and addressing threats. In the company, this is to be done by extended detection and response (XDR) tools and security information and event management (SIEM). The internal incident response plan again determines the response. The security team is responsible for the activities implemented based on this practice.

On top of all the practices implemented from the SSDF and MSDL, the critical asset needs to be backed up according to the internal backup policy, and that needs to be tested and updated regularly. The local legislature should also be checked to be compliant with data retention limits.

6 Found Hindrances during the Implementation

The biggest hindrance during the implementation of the SSDLC was very similar to many other companies – resources such as time and money, skilled personnel and the management's stance towards the need to secure the SDLC. The security manager is the only person except me who actively deals with security. That said, it is very difficult to incorporate all the best practices and suggestions for SSDLC, as only two people are not able to implement everything in a short time and also to monitor everything by themselves. There are, of course, other people in the company who have a basic or better understanding of cybersecurity, but it is not their main job in the company.

Connected to that is also the willingness or lack thereof of the employees, especially the developers, to change the way they work. Since there are many proposed changes to the SDLC as they know it, it will take them a longer time to go through the phases. That is even more true now at the beginning when they are just getting to know the new SSDLC. This can lead to capacity problems, the time estimates for new changes might be higher, and it is possible that the development of new functionalities will slow down. However, if this leads to a more secure product, one would argue that it is still worth it. The support of the senior management is vital here because otherwise, the changes will not be accepted easily.

As the company is small, the budget to incorporate security into the SDLC is also relatively small. That can be a problem mostly when considering different tools for different test types, such as SAST, DAST, etc. Many of the tools available are very expensive compared to what they would bring the company when chosen instead of the cheaper ones. That can also be because of the company's capacity and knowledge on how to correctly work with the results, meaning that if there were more people in the security team with different knowledge, perhaps the more expensive tools would show that they bring so much more than the cheaper ones. However, the current state limits the options to free or at least cheaper products that do not always offer all the possible functionalities.

The next hindrance was caused by my own lack of knowledge of the SDLC in the company. As I am not a developer, I had to consult many things with the development team to learn how exactly the whole SDLC works, what the company does regarding securing the SDLC and the product itself, etc. It was difficult mostly because the developers were and still are very busy with their planned tasks and also because, as I do not see deeply enough into the development, we had to meet more than once to specify some things and clear out some misunderstandings. However, all the developers were trying to help me as much as they could, and there was also one developer who was specifically assigned to help when needed.

Furthermore, incorporating the frameworks themselves was challenging. The whole SDLC needed to be somewhat changed, and for that, the internal processes also needed to change. That is not an easy task, even in a small company, as the processes should stay relatively unchanged for longer periods. The company is also in the process of changing the release process so there was no point in incorporating the policies into the processes yet. When this is finished and tested, the policies will be further incorporated into the according phases. These are the reasons why, now at the beginning, the SSDLC is managed mainly by the two new policies. First, when these show how well the people involved in the SSDLC can work with the new changes and the release process is updated, then the processes for new changes and SDLC can also be changed. This should happen in the following months.

During the analysis of the existing frameworks and methodologies, it was not always easy to find one with enough specifics. Even some of the ones analyzed for this thesis are not very detailed and, therefore, do not serve as a very good guideline for a company that is not experienced in this field. There are more solutions available, but many are also from lesser-known entities and are not easily adapted to any company with any development lifecycle. That proved to be a challenge, and in the end, only three frameworks were really considered for the implementation, as they offered enough details for successful implementation.

On top of the lack of detailed frameworks and methodologies, there was also a problem finding general publications regarding the implementation of an SSDLC. In the end, two publications and sev-

eral scientific articles were chosen for the thesis. However, the found frameworks and methodologies showed to be the best resources for the implementation in the end.

7 Possible Future Security Improvements

As already mentioned before, not every recommendation or practice of the frameworks could be incorporated into the SDLC at once. There is a possibility for incremental improvements with the frameworks, especially by implementing the practices that were left out during this first implementation. Some practices and tasks have already been implemented, but not up to their full content. This can also be improved in the next iterations.

The security personnel is limited in the company, so some things like the training cannot be done fully in-house at the time of writing the thesis. However, in the future, possible training for the security roles in the SSDLC should be prepared and overseen by the security team. These training plans are to be developed in the following months following the first implementation of SSDLC.

The policies themselves are only the first versions, and in the future, they might need revisions based on the recommendations collected during the first months of implementation. It is possible that some recommendations will, in the end, not be completely possible in the company and that some other practices can be added on top of them. Furthermore, the recommendations and practices from other frameworks can also be incorporated into the policies, when the ones described now are already fully incorporated in the SSDLC.

Further improvement can also be achieved by using the suggested Microsoft tools or further resources to the MSDL [30]. Some tools are already mentioned in the practices themselves, and some of these are already used in the company, but there is still a lot to be improved.

With the mention of specific tools, the used SAST, DAST or SCA tools should be further analyzed and assessed in order to find out if there are any better available. As mentioned above, the cost of these tools is a big factor in choosing them. However, with the growth of the company as well as its products, the tools will probably need to be changed to more sophisticated ones. Hopefully, by that time, the cost will not be such a big problem for the company.

Lastly, the SDLC does not only have to be improved by specific frameworks or methodologies. During the writing of the policies and thesis itself, a lot more recommendations were found that could be

incorporated in addition to the SSDF and MSDL. Most of them are based on other companies' experiences, so the company can benefit from learning from other's mistakes and not repeating them.

8 Conclusion

The thesis had two main goals – to choose an appropriate Secure SW Development Lifecycle (SSDLC) and to implement it in a selected company.

In the first chapter, the five frameworks and methodologies chosen to be compared for the thesis are described in detail. The chosen ones are the NIST: Secure Software Development Framework (SSDF), Microsoft Security Development Lifecycle (MSDL), Securosis: Putting Security Into DevOps, Juniper Networks Secure Development Lifecycle (JNSDL) and Cisco Secure Development Lifecycle (CSDL). These were chosen primarily because of their renown and also the reputation of their publisher.

The second chapter introduces the selected company itself. The organizational structure is mentioned mainly to show who deals with security in the company and how the security team, along with its manager, is incorporated into the company. Next, the whole development lifecycle of the products, as it was before the writing of this thesis, is described. The SDLC of a single new change is also described, as it is more detailed and managed by a separate process.

The next chapter analyses the security of the SDLC. Each phase is assessed for security flaws but also for what the company does well. For the analysis, the help of the developers was needed, as the security team (that I am a part of) was previously not very well integrated into the SDLC. In the end, the security strengths and flaws found are described in more detail independently of the phases in which they occur.

The first goal of the thesis, the process of choosing the best framework for the company, is described in the fourth chapter. Firstly, the focus points and metrics that were compared are described, and then the comparison of the frameworks is based on how well they cover the found security flaws. At the end of the chapter, the best framework blend is described.

The fifth chapter describes the implementation of the NIST SSDF and MSDL, so the second goal of the thesis. For that, two policies, the Secure Software Development Lifecycle Policy and Secure Coding Policy, were written and are attached to this thesis but also described in

this chapter. Each phase of the improved SSDLC is described together with the practices that could be implemented for it.

Next, the hindrances that were found during the implementation are described. These include some challenges caused by the company's lack of some resources, the state of the SDLC before the implementation of the thesis, the company's stance towards cybersecurity altogether, and the lack of my own experience with the topic and SDLC of the company.

The last chapter discusses some possible future improvements to the SSDLC. These can be further improvements based on the chosen frameworks, as not every task and practice could be implemented in the first round of implementation, but also improvements not suggested by the SSDF and MSDL. There is always room for improvement, and with cybersecurity, one can never think that what they have done is enough and does not need to be regularly assessed and updated. That is what will be done in the company as well, and this thesis, together with the policies, should now serve as a baseline and first iteration.

A Electronic Attachments

The following documents are electronically attached to this thesis:

1. **Secure_Software_Development_Lifecycle_Policy.pdf** – the policy describing the whole SSDLC
2. **Secure_Coding_Policy.pdf** – the policy describing in detail the phases of Implementation and Testing of the SSDLC

Bibliography

1. GUPTA, Sunil; GAYATHRI, N. Study of the Software Development Life Cycle and the Function of Testing. In: *2022 International Interdisciplinary Humanitarian Conference for Sustainability (IIHC)* [online]. 2022, pp. 1270–1275 [visited on 2024-09-04]. Available from DOI: 10.1109/IIHC55949.2022.10060231.
2. UMEUGO, Wisdom. SECURE SOFTWARE DEVELOPMENT LIFECYCLE: A CASE FOR ADOPTION IN SOFTWARE SMES. *International Journal of Advanced Research in Computer Science* [online]. 2023, vol. 14, pp. 5–12 [visited on 2024-12-09]. Available from DOI: 10.26483/ijarcs.v14i1.6949.
3. MOHAMMED, Nabil M.; NIAZI, Mahmood; ALSHAYEB, Mohammad; MAHMOOD, Sajjad. Exploring software security approaches in software development lifecycle: A systematic mapping study. *Computer Standards Interfaces* [online]. 2017, vol. 50, pp. 107–115 [visited on 2024-12-09]. ISSN 0920-5489. Available from DOI: https://doi.org/10.1016/j.csi.2016.10.001.
4. EIAN, Isaac; YONG, Lim; LI, Majesty; HASMADDI, Noor; TUZ ZAHRA, Fatima. *Integration of Security Modules in Software Development Lifecycle Phases* [online]. 2020. [visited on 2024-12-09]. Available from DOI: 10.48550/arXiv.2012.05540.
5. *NIST_SSDF.png* [online]. c2024. [visited on 2024-12-07]. Available from: https://blog.gitguardian.com/content/images/2024/09/5c95a2dede789e2bcc0a378ba5e54c51.png.
6. *Secure Software Development Framework | CSRC* [online]. [N.d.]. [visited on 2024-10-02]. Available from: https://csrc.nist.gov/Projects/ssdf.
7. Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities. In: [online]. [N.d.], p. 36 [visited on 2024-10-02]. Available from DOI: 10.6028/NIST.SP.800-218.
8. *About the Microsoft Security Development Lifecycle* [online]. c2024. [visited on 2024-09-18]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/about.

9. HOWARD, Michael; LIPNER, Steve. *The Security Development Lifecycle*. Vol. 34 [online]. 2006. [visited on 2024-12-09]. ISBN 0735622140. Available from DOI: 10.1007/s11623-010-0021-7.

10. *Zero Trust architecture and governance* [online]. c2024. [visited on 2024-12-04]. Available from: https://cdn-dynmedia-1.microsoft.com/is/image/microsoftcorp/sdl-graphic-fullscreen:VP5-1920x600.

11. *Security Program Management* [online]. c2024. [visited on 2024-09-30]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/security-program-management.

12. *Secure Platforms* [online]. c2024. [visited on 2024-09-30]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/secure-platforms.

13. *Secure By Design* [online]. c2024. [visited on 2024-09-30]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/secure-by-design.

14. *Cryptography* [online]. c2024. [visited on 2024-10-02]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/cryptography.

15. *SSCS* [online]. c2024. [visited on 2024-10-02]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/sscs.

16. *Secure Dev Infrastructure* [online]. c2024. [visited on 2024-10-02]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/secure-dev-infra.

17. *Security Testing* [online]. c2024. [visited on 2024-10-02]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/security-testing.

18. *Operational Security* [online]. c2024. [visited on 2024-10-03]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/operational-security.

19. *Monitoring And Response* [online]. c2024. [visited on 2024-08-20]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/monitoring-and-response.

20. *Security Training* [online]. c2024. [visited on 2024-08-20]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/practices/security-training.

21. *Putting Security Into DevOps* [online]. c2023. [visited on 2024-10-05]. Available from: https://cdn.securosis.com/assets/library/reports/Security_Into_DevOps_Final.pdf.

22. *Juniper Networks Secure Development Lifecycle* [online]. c1999–2024. [visited on 2024-08-30]. Available from: https://www.juniper.net/content/dam/www/assets/white-papers/us/en/juniper-networks-secure-development-lifecycle.pdf.

23. *Cisco Secure Development Lifecycle* [online]. c2024. [visited on 2024-12-14]. Available from: https://www.cisco.com/c/dam/en/us/products/collateral/customer-collaboration/unified-contact-center-enterprise/white-paper-c11-740413.docx/_jcr_content/renditions/white-paper-c11-740413_0.png.

24. *Cisco Secure Development Lifecycle Overview* [online]. c2024. [visited on 2024-08-17]. Available from: https://www.thestack.technology/content/files/c/dam/en_us/about/doing_business/trust-center/docs/cisco-secure-development-lifecycle.pdf.

25. UMEUGO, Wisdom. SECURE SOFTWARE DEVELOPMENT LIFECYCLE: A CASE FOR ADOPTION IN SOFTWARE SMES. *International Journal of Advanced Research in Computer Science* [online]. 2023, vol. 14, pp. 5–12 [visited on 2024-10-08]. Available from DOI: 10.26483/ijarcs.v14i1.6949.

26. SULTAN, Khalid; EN-NOUAARY, Abdeslam; HAMOU-LHADJ, Abdelwahab. Catalog of Metrics for Assessing Security Risks of Software throughout the Software Development Life Cycle. In: *2008 International Conference on Information Security and Assurance (isa 2008)* [online]. 2008, pp. 461–465 [visited on 2024-09-24]. Available from DOI: 10.1109/ISA.2008.104.

27. Chapter 5: Application Security and ISO27001. In: *Application Security in the ISO 27001:2013 Environment*. Second edition. IT Governance publishing, 2015, pp. 43–90. ISBN 978-1-84928-767-8.

28. MEKRUKSAVANICH, Sakorn; JITPATTANAKUL, Anuchit; THONGKUM, Patcharapan. Metrics-based Knowledge Analysis in Software Design for Web-based Application Security Protection. In: *2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering* [online]. 2021, pp. 281–284 [visited on 2024-10-17]. Available from DOI: 10.1109/ECTIDAMTNCON51128.2021.9425703.

29. ASSAL, Hala; CHIASSON, Sonia. Security in the software development lifecycle. In: *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security* [online]. Baltimore, MD, USA: USENIX Association, 2018, pp. 281–296 [visited on 2024-12-09]. SOUPS '18. ISBN 9781931971454.

30. *Microsoft Security Development Lifecycle Resources* [online]. c2024. [visited on 2024-12-07]. Available from: https://www.microsoft.com/en-us/securityengineering/sdl/resources?oneroute=true.