Mike Vainio

Practical Framework for Continuous Security

Metropolia University of Applied Sciences

Master's Degree

Degree Programme in Business Informatics

Master's Thesis

20.11.2023

Abstract

Author:	Mike Vainio
Title:	Practical Framework for Continuous Security
Number of Pages:	77 pages + 1 appendix
Date:	20 November 2023
Degree:	Master of Business Administration
Degree Programme:	Business Informatics
Instructor:	Antti Hovi, Senior Lecturer

This thesis seeks to identify a structured way for integrating security seamlessly with a DevOps driven software development and delivery process. Instead of relying on specialised security personnel, this thesis describes methods and practices that aim to be easy adopt and actionable for anyone working within the software development or delivery process.

This thesis used applied action research as its research approach. The research design consisted of five steps and three data collection rounds. The first data collection round was for the current state analysis of the case company's security processes and practices, it consisted of one-to-one interviews, workshops, and a survey. The second data collection round was for proposal building, it consisted of peer-reviews, interviews and a workshop. The third data collection round for validation of the initial proposal, it consisted of a workshop.

The current state analysis describes and analyses the case company's existing security processes and practices. The findings from the current state analysis helped to identify areas for the literature and best practice search that resulted in building the conceptual framework. The findings from both current state analysis and existing knowledge helped to shape the initial proposal in the next stage, proposal building.

The proposal was built in three steps. First, initial scoping and planning was done in small workshops to find a suitable structured approach to security that is relevant in the case company's context. Second, the initial materials were distilled into a draft of the proposal, and feedback and suggestions on the draft was gathered. Third, with the feedback from the second step, the draft of the proposal was updated and more practical information and examples were developed into the proposal.

As the outcome, the thesis demonstrated how security can be integrated into a DevOps driven software development and delivery process via a security framework. The identified structured way for integrating security practices into the DevOps driven process provides both high level strategic guidance for effective communication and low level practical guidance for actionable steps. The practical implementation of the structured approach lays a foundation for secure implementations of future projects.

Keywords: Cybersecurity, Security, DevSecOps, Continuous Delivery

Contents

Glossary
List of Figures

1 Introduction 1

1.1 Business Context 1
1.2 Business Challenge, Objective and Outcome 2
1.3 Thesis Outline 2

2 Method and Material 3

2.1 Research Approach 3
2.2 Research Design 4
2.3 Data Collection and Analysis 5

3 Current State Analysis of Security Practices and Processes 9

3.1 Overview of the Current State Analysis 9
3.2 Description of Current State of Security Practices and Processes 10
3.3 Analysis of Current State of Security Practices and Processes 12
3.3.1 Software Security Lifecycle Phases 12
3.4 Key Findings from the Current State Analysis 19
3.4.1 Strengths and Weaknesses of the Current Security Practices and Processes 19
3.4.2 Selected Key Areas for Improvements 20

4 Existing Knowledge on Securing Software Modern Software Delivery 22

4.1 DevSecOps 22
4.1.1 DevOps Driven Software Delivery Process 22
4.1.2 Continuous Integration and Continuous Delivery 23
4.1.3 DevSecOps 26
4.1.4 Shift Left Security 26
4.1.5 Test Driven Security 27
4.2 Software Security Lifecycle Phases 27
4.2.1 Develop 30
4.2.2 Distribute 30
4.2.3 Deploy 31

4.2.4 Runtime 31
4.3 Software Supply Chain Security 32
4.3.1 Signing 36
4.3.2 SLSA 37
4.3.3 SBOM 38
4.4 Secrets Management 39
4.4.1 Secrets Sprawl 40
4.4.2 Secrets Managers 41
4.5 Securing Source Code 41
4.5.1 Access Control 42
4.5.2 Monitoring 43
4.6 Conceptual Framework 44
5 Building Proposal for Continuous Security Framework for the Company 47
5.1 Overview of the Proposal Building Stage 47
5.2 Findings from Data Collection 2 48
5.3 Initial Proposal 51
5.3.1 Element 1 of the Initial Proposal: DevSecOps Culture and Methods 53
5.3.2 Element 2 of the Initial Proposal: Secure Software Development Lifecycle Phases 55
5.3.3 Element 3 of the Initial Proposal: Foundational Security Practices 59
5.3.4 Element 4 of the Initial Proposal: Security Landscape 61
6 Validation of the Proposal 64
6.1 Overview of the Validation Stage 64
6.2 Developments to the Initial Proposal 65
6.2.1 Developments to Element 1: DevSecOps Culture and Methods 67
6.2.2 Developments to Elements 2: Secure Software Development Lifecycle Phases67
6.2.3 Developments to Elements 3: Foundational Security Practices 69
6.2.4 Developments to Element 4: Security Landscape 69
6.3 Final Proposal 70
6.4 Implementation 72
7 Conclusion 74
7.1 Executive Summary 74
7.2 Thesis Evaluation 76
7.3 Closing Words 77

References 78

Appendices

Appendix 1. Current State Analysis Survey

Glossary

CI	Continuous Integration. Practice of testing changes from multiple developers to find issues and conflicts as soon as possible. Term continuously means it is done often, ideally multiple times a day.
CD	Continuous Delivery. Term used to describe a way of working where application code in a repository is always ready to be delivered, in other words every addition to the code base (commit) is a potential release.
IaC	Infrastructure as Code. Term used with tools that allow describing infrastructure in text files, when the tool is executed it creates the corresponding infrastructure resources.
SCM	Source Control Management. System designed for storing and managing source code, synonym for Version Control System.
VCS	Version Control System. Synonym for SCM, system designed for storing and managing source code.

List of Figures

Figure 1. Research design of this study.....4
Figure 2. Example Deployment Pipeline, re-drawn. (Continuous Delivery, p 3-4, 106). 25
Figure 3. Re-drawn diagram of the Lifecycle Phases (CNCF 2022, 10, 12, 18).....29
Figure 4. Runtime Lifecycle Phase (CNCF 2022, 19).....32
Figure 5. Diagram of the amount of observed Software Supply Chain attacks between 2019-2023 (Sonatype 2023).....34
Figure 6. Supply Chain Threats. (SLSA.dev 2023).....35
Figure 7. Software Bill of Materials visualized along with baseline software component information (Figure 1, NTIA 2020).....38
Figure 8. Conceptual Framework for a security framework. 45
Figure 9. Enhanced Deployment Pipeline.....54
Figure 10. Continuous Integration and Continuous Delivery scoped in the Deployment Pipeline 54
Figure 11. Security Landscape 62
Figure 12. View of the Software Security Lifecycle Phases on the company´s website. 72
Figure 13. Navigation bar of the website. 73

1 Introduction

With the ever-increasing threat of cyberattacks and data breaches, organizations must start to take security more seriously than ever. Most organizations and individuals rely on digitalized solutions in their daily operations that makes their life easier, or their business prosper. Many also software applications also deal with and store sensitive data, which must be protected from unauthorised access or exfiltration. Thanks to advances technology such as *ad hoc* infrastructure available from Cloud Providers such as AWS, and new methodologies in software development such as Agile and more recently DevOps, applications are being deployed and evolved faster than ever and new features make it to the hands of the users faster than ever. These users trust the application vendors to deliver a secure and reliable experience, keeping the users credit card and other personal information private.

Unfortunately, the increased speed in the software industry development and delivery cycle causes significant challenges for security experts and their existing processes. High performing development teams are deploying new releases of their software multiple times per day, and there's no time for a security audit when the deployments are fully automated. (Forsgren et al. 2018, 42-43) It means, security practices and processes must evolve to meet the needs of these high performing teams. IT professionals in various roles should bake in security for all the processes and operations, however not everyone has the experience and skills required to implement secure solutions on their own. (Vehen 2018, 2-3)

This thesis studies how security practices can be integrated into the development and delivery workflow of high performing teams practicing DevOps.

1.1 Business Context

Verifa is a small IT consultancy focusing on cloud architecture and continuous software development and delivery processes. Instead of writing the software application itself, Verifa consultants aid with the process of continuously integrating and delivering that software. Helping teams reach their full potential and unlock bottlenecks in the process. The software delivery process is meant to be largely automated with today's industry standards and security should be considered in every step of the way from the source

code repository to the final deployment to a public cloud platform or a private cloud. Verifa consultants need up-to-date and actionable knowledge on how to integrate security into their consulting projects.

1.2 Business Challenge, Objective and Outcome

The business challenge is to identify right tools and practices that enhance the security awareness and knowledge of the consultants, so that consultants know what security practices to implement when working in customer projects. Also, consultants need this knowledge to gain confidence that they can have the right approach, tools, and skills to ensure security is baked into every customer delivery. Thus, the objective of this thesis is to *create a practical continuous security framework which would equip consultants with the tools for effective customer communication and effective security practices for implementing projects.*

The outcome of the thesis is *the first iteration of the continuous security framework which can be taken into use in internal and customer projects.* Creation of the framework should also raise awareness around security and help with producing related marketing material, such as blog posts. This first iteration can then be further evolved within the company, guided by the feedback from various projects.

1.3 Thesis Outline

The scope of this thesis is to create a continuous security framework for all the consultants of the case company since security affects every are of IT, although consultants have with different specialisations and backgrounds. This thesis is conducted by mixing implementation, data and feedback collection, and literature research on existing knowledge.

This thesis contains seven sections. Section 1 is the introduction to the topic and the business context. Section 2 describes the methods and materials used in this thesis. Section 3 dives into the current state of security practices and processes of the case company. Section 4 dives into literature on selected focus areas from the current state analysis. Section 5 describes the initial proposal of the continuous security framework. Section 6 is dedicated to testing and validating the initial proposal. Finally, Section 7 in for the executive summary and the closure of the thesis.

2 Method and Material

This section describes the research methods, research design and data collection methods used in this Thesis.

2.1 Research Approach

The choice of research approach and methods depends on the research problem. (Kananen 2013, 28). The two most popular *research families* are quantitative research and qualitative research. Quantitative research, as the name suggest, deals with statistical analysis of numerical data. (Adams et al. 2013, 6) However, research approaches are often a mix of quantitative and qualitative research methods.

Common *research strategies* include case studies and action research, which both utilise quantitative and qualitative research methods. Action research aims at finding a practical solution to a real business problem via change, whereas case study aims to merely observe the phenomenon and produce knowledge. (Kananen 2013, 28-29, 37-39) Applied Action Research (also known as Design Research) aims to research a business problem by carrying on a development work and then researching that in the context of the case company, without trying to generalise the results. Result from a such a study can be a new product, service, concept or improved issues thanks to the change.

In this thesis, qualitative research methods are the main methods used in this Thesis. Some quantitative elements are also used to analyse the numerical data from the data collection, for example, data from a survey, but the majority of the methods used are qualitative research methods and data gathered from interviews, workshops, along with the analysis of internal documents and systems. Thus, Applied action research (also known as Design Research) is used as the research approach of this Thesis. It fits well into a thesis objective and a single iteration of its research design versus multiple iterations such as in Action Research. The thesis aims to combine research and development, and to produce a solution that can be evaluated by presenting the implementation results and gathering feedback from different stakeholders for further development. The solution is a security framework, and the author is involved in the building of this framework working together with peer consultants of the case company.

2.2 Research Design

The research design of this thesis is a single iteration research and development process split into five steps where each step has an input and output. Three of the steps in the design include data collection that is then analysed during the step and affects their outputs. Figure 1 below shows the research design of this study.

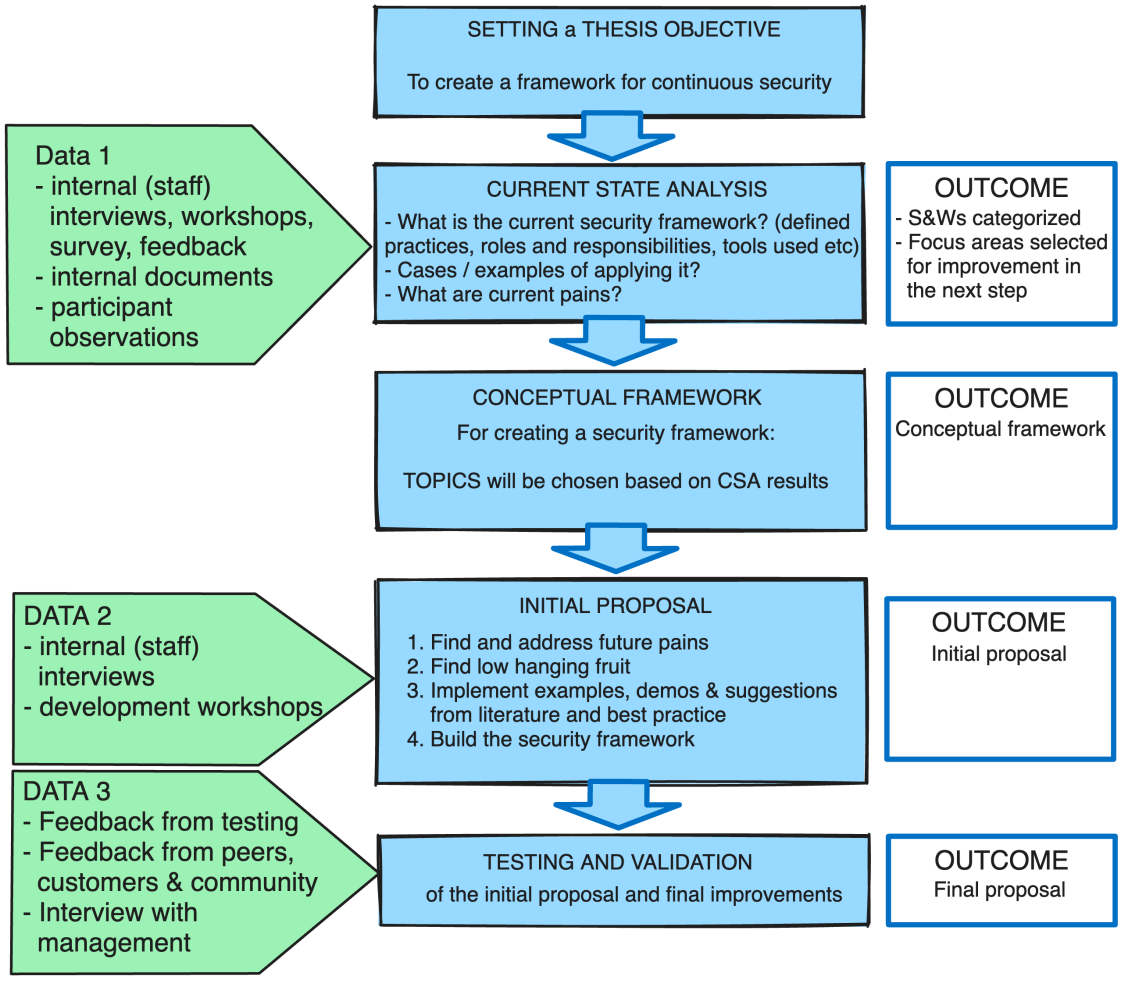

Figure 1. Research design of this study.

As shown in Figure 1, the design starts with setting the objective, which is creating a framework for continuous security. The second step is the current state analysis which is supported by data collected from internal staff interviews, workshops, and a survey. In the current state analysis, the collected data is analysed, and the outcome of the current state analysis are the strength and weaknesses and focus areas that feed into the conceptual framework.

Step three is the conceptual framework where existing knowledge around the identified focus areas is explored to find and identify suitable industry best practices and patterns that could be utilised in the initial proposal. Outcome of the step three is the conceptual framework.

Step four is the initial proposal which is again supported by data collected from internal staff in the form of interviews, conversations, and development workshops. This data is used for early feedback to guide the proposal design and implementation. Focus for the initial proposal is to address the pains identified during current state analysis, address low hanging fruits and implement the initial version of the security framework. Outcome of the step four is the initial proposal.

Step five is the validation and testing of the initial proposal. Step five is again supported by data which is collected from the internal testing of the proposal and from interviews with peers and management, and internal workshops. Outcome of the step five is the final proposal.

2.3 Data Collection and Analysis

This study draws from a variety of data sources, the data was collected in several iterations over a time period of almost a year between fall of 2022 and fall of 2023. Table 1 shows the details of Data collection 1-3 used in this study.

Table 1. Details of data collection 1-3 used in this study.

Participants / role	Data type	Topic, description	Date, length	Documented as
Data 1, Current state analysis / scoping				
Internal stakeholders	Workshop (face-to-face and online)	Current security practices. Core security competences and weaknesses.	Q3 2022, 1-1.5 hour sessions.	Field notes. Interactive canvas (excalidraw)
Peer Cloud Architects	Interview (face-to-face and online)	Current security practices. Core security competences and weaknesses.	Q1 2023, 30 minutes.	Recordings (online), field notes (face-to-face)
Internal stakeholders	Survey	Core security competences and areas for improvement.	Q1 2023	Survey analysis notes
Data 2, for Proposal building (action)				
Internal stakeholders	Demo	Overall structure of the continuous security framework.	Q1 2023, 1h	Field notes. Interactive canvas (excalidraw)
Peer Cloud Architects	Peer review	Implementation details on different elements.	Q1-Q2 2023, 15-30 minutes	Field notes
Peer Cloud Architects	Interview	Suggestions and comments on scope of the continuous security framework.	Q1-Q2 2023, 5-15 minutes.	Field notes
Data 3, from Validation				
Internal stakeholders	Workshop (face-to-face)	Feedback on initial implementation. Suggestions for improvements and clarifications to the continuous security framework.	Q3 2023, 1.5 hours.	Field notes

As seen in Table 1, the data for the study was collected in three rounds: Data 1, Data 2, and Data 3. Data 1 was collected for the current state analysis step, and Data 2 for the proposal building step and finally Data 3 for the for the validation, testing and building of the final proposal.

In the first round, Data 1 was collected to gain a deep understanding of the current security processes and practices of the case company. The data was gathered using a combination of interviews, workshops, and a survey. First, an online workshop was held where the idea for creating a security framework was introduced to everyone present, all case company consultants were invited to the workshop. Second, based on the conversations in the initial workshop, few people were selected or individual interviews as they had interest towards security and insights about current the security practices and processes. The interviews were held both online and face-to-face, because of different office locations face-to-face was not possible with everyone. Finally, a survey

was created and everyone in the case company was invited to take time to answer the survey. The questions for the survey can be found in Appendix 1.

In the second round, Data 2 was collected to gather feedback on the implementation, design, and scope of the proposal. The data was gathered using a combination of free-form interviews, peer-reviews and a short workshop with technical consultants of the case company. First, a workshop was held to get everyone's opinions on the overall structure of the continuous security framework and what the different elements would be. Second, free-form peer reviews happened with consultants from the same office during short co-working sessions with peer consultants that were interested in the topic. Finally, free-form conversations that influenced the structure of the framework happened organically at the office during the proposal building. All the feedback and suggestions for Data 2 were gathered as field notes and some visual ideas for the structure captured in an interactive canvas.

In the third round, Data 3 was collected to validate and test the initial proposal. Based on feedback from this data collection, the final proposal was built. The data was gathered in a workshop where the implementation of the initial proposal was presented and then free-form feedback from peers was recorded as field notes. The Data 3 was collected during a yearly company conference with most of the company staff present. The continuous security framework was voted as one of the topics that would be in the agenda as a workshop session. During the workshop a presentation was held that consisted of demonstrating the initial proposal as the initial website implementation, after the demonstration a free form feedback session was held when participants raised various points of feedback and suggestions for next steps. The feedback was gathered as field notes.

The interviews were carried out both online via video conferencing and face-to-face at the company premises. The interview questions were prepared beforehand, and the online interviews were recorded while field notes were made of the face-to-face interviews to record the conversation. Also, workshops were held both online via video conferencing and face-to-face. Workshops were always recorded as field notes. Finally, Data 1 also included a survey which consisted of ultimately open-ended questions, but some suggestions and ready-made example choices were given to the participants.

The biggest data collection effort and analysis effort was done in the first step for the current state analysis to gain a deep understanding of the state of the current security processes and practices of the case company. The findings of the current state analysis are discussed in Section 3 below.

3 Current State Analysis of Security Practices and Processes

This section of the thesis analyses the current state of security practices and processes at the case company. The description of current Security Practices and Processes establishes the current way of working with regards to security in software development, delivery, and operations of the case company. The analysis seeks to understand the problems with the current ways of working and then find out what could be improved and what is missing in order to enhance the security posture of all internal and customer projects.

3.1 Overview of the Current State Analysis

The goal of the current state analysis is to establish the current state of the security practices and processes, roles and responsibilities and tools used. To gather the data for the analysis, internal interviews and workshops were conducted between January and June 2022. Both internal and external projects (customer work) were discussed during the interviews and workshops to identify common pains and frustrations in current way of working.

First, interviews were carried out partly face to face and partly in an online meeting due to the company employees locating in different countries. The online meetings were recorded, and field notes were taken for both type of interviews. Second, a workshop was conducted face-to-face in an internal conference with almost all employees present in the workshop. The input from the workshop was written down as field notes interactively during the session. Third, several short “opt-in” workshops were conducted online with participants that were interested and available to join the discussion. The input from the workshop was summarised in field notes and the meetings were recorded for later reference. Finally, a survey was conducted to which half of the consultants answered. The answers and statistics of the survey were stored in the survey tool itself. The survey was intentionally the last data gathering phase so that the questions would be informed by the data already gathered.

The current state analysis is split into two parts. The first part describes the current practices backed by the data gathered and the second part analyses which development areas were identified given the current state.

3.2 Description of Current State of Security Practices and Processes

The case company offers consultancy services for customer organisations that create software applications, instead of writing those the code for those applications, the consultants support that process in various ways. This can range from workshops with Value Stream Mapping exercises to hands-on coaching and implementation work to setup a tool that a team needs. Often some amount of strategical coaching and consulting is needed to truly discover the bottlenecks and waste in the current way teams of the customer organisation are working and build a road map before any hands-on implementation can be started.

The case company consultants naturally have different backgrounds and specialisations, but ultimately though, the technical aspects of a most customer projects consist of similar concepts although tools and technologies would be different. Projects can be split roughly two categories: Cloud and DevOps. The Cloud category consists of projects where a customer is looking migrate into a cloud platform or improve their existing processes and architecture around the cloud platform. The DevOps category is broader, customer organisation might be struggling to adopt and evangelize a DevOps strategy and culture in their organisation or customer organisation might need help with the technical aspects of setting up new tools and driving user-adoption of those solutions. Between the two categories there is also often overlap, many customer organisations have realised that operating in a cloud platform with a lot of “as a Service” tooling available, can provide a much-needed speed and agility to operations that can help to foster a DevOps culture as opposed to operating in on premises data centres.

When it comes to technical, hands-on and architectural work the case company consultants do need to work with security related concerns, but it’s rarely the main value that a customer organisation is seeking from the delivery. Instead, customers are interested in shipping higher quality software, faster and only last, securely. Security has a value, but security done right is often transparent and not obvious that it’s done as part of a larger design and implementation. Also, often specialised experts from the customer organisation are brought in close to the end of an implementation project to audit or review the security aspects of the design or implemented solution. Consultants work in senior roles with customer organisation peers, such as Cloud Architects or Senior Cloud Engineers and are there to guide the organisation on best practices in the technical and cultural areas of Cloud and DevOps. Whilst security is present in the daily work for many

of the consultants, it's not obvious how it should be included in customer work and how security integrates with a DevOps strategy implemented on top of a cloud platform.

Next, some findings from the data collection from internal stakeholders are discussed, diving deeper into the daily tasks of different consultants.

The interviews were done with consultants working on different projects, this was intentional to gather a wide view of the current practices and processes, including various customers and their requirements.

In the interviews, one of the questions was about how seriously we take security into account in our customer work, in one of the interviews a consultants shared the following:

> "We talk about security like it's important, but in reality we have no established way of doing security. I feel security could follow a common model for all customer projects." (Consultant 1)

The consultant is referencing to the fact that internally there is documentation or guidance about security practices. But this does not mean that security best practices are not followed or customer solutions would be insecure, it is just not well defined within the organisation what those practices are. Another interview questions was do we explicitly include security in the design and implementation for customer projects. The same interviewed consultant replied:

> "It depends fully on the consultant. We do some basics like firewall configurations consistently, but not much more than that." (Consultant 1)

In a workshop, when talking about how our customers approach security we could see a common trend of companies trying to "shift left" with security, meaning security scans and other activities would happen earlier in the software development process. However, these initiatives were just known for the consultants, and they had no actual impact in their workflow during development. At the same time in the interviews, it was clear that customers deeply care about security and see it critical for business continuity. Where these initiatives do show is when a centralised security team does scanning of infrastructure and application landscape.

Overall, we could quickly derive that our consultants don’t work on directly security related tasks other than very rarely or when it’s triggered by an event/request external to the development team that the consultant is in (for example a mandatory review before going into production with a new service). Security is often an afterthought which is dealt with if time and resources allow, and consultants don’t have a concrete view of the security posture, nor do they get continuously feedback during development on security posture of the code.

During the workshops it was also discussed if security tooling is something that consultants want to work with, and the answer was purely positive, this is something that should be paid more attention to. In an interview one consultant shared his view on what kind of impact that might have as well:

> “Yes, it's like CI [Continuous Integration] - first you think who needs that, then you do it and you start to long for it.” (Consultant 2)

Although it was clear that security is a key non-functional requirement for any project and many consultants expressed, that they feel they don’t have the necessary knowledge. And the survey results show that security tasks either don’t exist or they have low priority.

3.3 Analysis of Current State of Security Practices and Processes

This analysis categorizes the findings from the data gathered during interviews, workshops, and the survey. The analysis tries to point out pain points and potential points of improvement.

3.3.1 Software Security Lifecycle Phases

In order to structure the findings of the most technical parts of the current state analysis, the lifecycle of modern software development, operations and maintenance is split into lifecycle phases and foundational security practices. In this thesis, the lifecycle phases are presented as a modified version of the lifecycle phases described in the CNCF Cloud Native Security Whitepaper, the original lifecycle phases from the whitepaper are examined in the existing knowledge chapter, and the modified model is explained in the proposal chapter.

The four lifecycle phases are: (1) Develop and Integrate, (2) Test and Distribute, (3) Deploy, and (4) Monitor. The current state analysis findings will be discussed next under each of these categories to present and analyze them more clearly in the structure of several upcoming sections of the thesis.

3.3.1.1 Develop and Integrate

As the consultants of the case company don't write application code, the findings in the CSA are related to other type supporting code and configuration that does not directly go into the running software products of the customers. The case company consultant development consists largely of Infrastructure as Code (IaC) configuration, internal tool development and Continuous Integration (CI) and Continuous Delivery (CD) pipelines.

In a workshop conducted by most of the case company consultants present, a consultant shared an experience of using a tool for static analysis of Terraform configuration using an open-source tool.

> "I run a scan with tfsec in my previous project for the IaC I had implemented for them and the results from the scan were useful, however I only did it once" (Consultant 3)

This example shows that tools that are easy to run and available for free are very useful when they are run. But unfortunately, they are not run continuously which means the feedback might come too late and refactoring of the configuration is needed after work is considered finished, instead of during the work, which causes rework. Then, there's a need to go back and change the configuration because a security flaw is discovered.

In a workshop, there was a discussion on what existing practices the consultants practice what is important for security. Surprisingly it was nothing directly security related that was found. Rather it was discovered that working in a certain way enables the use of security tools that can do static analysis. All the consultants are committed to using best practices for the technical aspects of DevOps and Continuous Delivery, which means a high level of automation and "Everything as Code" is the typical goal.

This was an important finding because it provides a firm foundation for the static analysis tools, such as tfsec, to be used to automatically scan all the work done by the consultants. When developing something new in small, iterative steps, a tool such as tfsec could be introduced in the Continuous Integration pipeline. It is run on whenever a commit is pushed to the Version Control System, providing timely and continuous feedback to the developer and others working with the same codebase. If consultants were not already practicing these best practices, it would be impossible to use tools early in the process, and instead most likely the runtime environment would have to be scanned after the infrastructure running and possibly already in a vulnerable state.

3.3.1.2 Test and Distribute

Between “Develop” and “Build and Distribute” phases, the scans might be very similar as what can be checked directly when the code is developed if everything is written “as code”. However, now it is also possible to build the project and run more long running tests. During the interviews and workshops with the consultant, it was noticed that there’s very little security related tests that they write, thus there was very little data gathered about this topic. Instead of writing security related tests, most enterprise customers use Commercial Off-The-Shelf (COTS) products that are integrated in the build and test pipelines, or simply run on schedule. Some of the example tools discovered were: (1) Black Duck, (2) SonarQube, and (3) JFrog Xray.

These tools check either the source code directly or the produced artifacts, naturally some of these tools are also integrated in the Continuous Integration pipeline, but that’s not possible if they scan produced artifacts from a build. Setting up these integrations with tools is often done by the case company consultants.

In the interview with a consultant, he brought up that the customer is looking into using an off-the-self solution to scan all container images and other software artifacts. One of the benefits of using this solution is that it goes to work when built artifacts are pushed into the system that holds those artifacts. This means there’s not much coordination needed to enable this integration for different products as everyone is already using the artifact storage system. This seamless integration with the existing workflow is seen as an extremely useful perk of the system.

It was also discovered from the discussion that metadata from the build environment and the artifacts produced are becoming increasingly important, and new steps are needed to produce this sort of metadata.

Consultants of the case company are often very involved with the build steps, but still they found that rarely they are responsible to implement security controls when pulling in third party dependencies and introducing something like signing or producing SBOMs (Software Bill of Material). However, lately there has been more demand from customers and from the industry in software supply chain security, and many consultants see it as a mandatory addition to their work in the near future.

During the workshop conducted with most of the case company consultants present, a consultant shared that recently a new requirement was needed in the customer project he was working in. The customer had a need to cryptographically sign all the artifacts that go into their software product and in addition to signing they were also looking to implement Software Bill of Material (SBOM) document to be created whenever a releasable software artifact is built. This example shows that the software supply chain security practices are starting to show up in customer projects, but consultants still largely lack knowledge of how to approach these requirements in practice.

3.3.1.3 Deploy

During the deploy phase, the goal is to run pre-flight checks that make sure the deployment configuration is secure. During discussions in workshops and interviews, it was identified that there are two types of “pre-flight” checks that consultants had experience with:

1. Kubernetes Admission Controllers

2. Built-in policy and compliance tools in Cloud platforms.

Kubernetes itself provides many platforms features you might expect from a cloud provider, and thus it’s very popular to deploy an admission controller on a Kubernetes cluster that can check deployment configuration for misconfigurations or insecure defaults. Three consultants said they have worked with Kubernetes admission

controllers in the past and found them useful, although sometimes the writing the rules proved challenging since the languages used for defining the rules are not that well known to the consultants.

When consultants have been working in a public cloud environment, there might also be some policies in the cloud platform that are used to enforce certain guardrails to avoid insecure configurations. Defining these policies is not that common for the case company consultants, instead re-usable and safe modules of configuration/code are more common. However, having guardrails is seen useful and positive by most consultants.

Both the Kubernetes admission controllers and the cloud platform provided policy tools are great in theory, but consultants expressed frustrations especially with one customer that has a very broad set of policies set in their cloud platform subscriptions. When working with these policies in the cloud platform it becomes very cumbersome to work as the consultants are used to with other customers, as many things have to be done either through raising a ticket or going through a custom self-service portal/API. During the interview, a consultant working with this particularly policy-enforcing client expressed their concern:

> “Whenever there is friction, then the users are going to take a shortcut in skip the process” (Consultant 2)

The consultants had seen many situations where these security guardrails prevent daily work, if the policies get in the way of working. In this situation, the development teams will look for alternative ways of achieving their goals and often succeed to shortcut the process and end up lowering the security posture of the product or system.

3.3.1.4 Monitor

In the monitor phase, the solutions built by consultants are running in a cloud or on-premises environment and they must be continuously monitored for unexpected behaviour that might be caused by a threat actor targeting the solution.

Because consultants are rarely involved building and improving the monitor phase, the findings from consultants are that every now and then they would receive feedback from

a centralized security team that some solution in the runtime environment has been scanned or otherwise analyzed and what were the security findings. From the workshop conducted with most of the consultants of the case company present, a consultant shared the following experience:

> “Security experts conducted an audit on the running system [after it had been designed and implemented fully] and handed over a report with very useful finding and suggestions for improving the security” (Consultant 4)

Clearly the security experts and the tools used did a great job and the feedback was great. But the fact that the feedback came after the design and implementation was done could have caused a substantial amount of re-work to alter the design and implementation to fix the found security implications. In this case this was not the case, only minor changes were needed which could be implemented quickly. The consultant who shared the experience wished these audits would happen more frequently and earlier in the design and implementation process. The automated parts of this audit could perhaps be introduced in the CI/CD pipeline during “Test and Distribute” phase of the lifecycle.

During the day-to-day activities in another customer project, a few of the consultants from the case company were struggling with a similar audit and automated scanner process which was mandatory for the solution they were responsible for building and maintaining. In this case, the main issue was that this solution was a Commercial off-the-shelf (COTS) product. All that the consultants could do would be to find ways to mitigate the found issues if it was to be done at the load balancer or firewall level and report the remaining issues to the software vendor. This audit was largely not relevant for the consultants and better use of time would have been for the security findings to be reported directly to the software vendor. In other words, a large portion of the time spent discussing and analysing the findings was wasted, no meaningful improvements could be made to the security posture.

These two wildly different experiences show that perhaps it would be better for the team responsible for the solution/system to be more in charge the process of continuous security scanning.

3.3.1.5 Foundational Practices

The results from the internal survey show that topics that fit into the foundational practices are the ones that consultants find most important. In the survey, there was a question about which security practice is the most important, with 11 different practices listed covering all the phases of the secure software lifecycle and foundational practices. The four most popular answers were all in the foundational practices category:

- Secrets Management (6 votes)
- Supply Chain Security (3 votes)
- Version Control System Security (3 votes)
- Peer Review (3 votes).

As the results show, secrets management is seen as the most important foundational practice. In an interview with one of the consultants, secrets management also was brought up when discussing customers´ security practices. A consultant shared an experience where he has experienced secrets shared in plain text over chat applications (such as Microsoft Teams). Of course, if there's no dedicated tool for sharing secrets, this is understandable as work must get done, but this could be improved certainly.

Software Supply Chain Security, Version Control System (VCS) security and peer review were all seen equally important according to the survey results. In the survey, the consultants were also asked which of these practices were most actionable. The results show that VCS security and peer reviews are seen as easily actionable, but software supply chain security is more difficult as consultants are not aware what could be done. Secrets management is also a bit tricky because it in practice always requires changes to everyone's workflow and to get them to use a new tool, this was a challenge with one customer. Many of the other consultants also agree that one of the biggest challenges of improving a foundational security practice is that it often involves changing existing workflows and way of working. The bigger the change, the more difficult this naturally is. Finding the ways that have the least friction with existing workflow seems crucial.

3.4 Key Findings from the Current State Analysis

The findings from the current state are summarised below as strengths and weaknesses identified, and then focus areas are selected for further studying to find improvements in those areas.

3.4.1 Strengths and Weaknesses of the Current Security Practices and Processes

From the data and analysis of it, several key strengths and weaknesses were identified:

Strengths

1. Infrastructure as Code is valuable tool to quickly identify and fix security issues with infrastructure.

2. Strong competence in CI/CD makes it easy to start implementing additional security steps in pipelines.

3. Growth mindset, consultants are comfortable to learn the new tools and start implementing new type of work (security).

4. Everyone cares about protecting secrets and tries to do their best at it, we also have people with expertise on some popular secret management tools.

Weaknesses

1. Lack of experience with security tools and difficulties identifying suitable tools to recommend and implement.

2. No security framework for what practices to adopt and at what point of the lifecycle phases.

3. Security work is done ad-hoc instead of continuously, often triggered by an external team either by automated scan or full security audit.

3.4.2 Selected Key Areas for Improvements

A continuous security framework is needed for consultants to have clear structure for implementing security and communicating effectively with customers. Adopting continuous practices will enhance the feedback loop from writing code to knowing the security posture of it. Consultants need additional upskilling in security tooling and practices, which makes it easier to integrate security tooling into their customer work. Especially some foundational security practices that many of the consultants found interesting should be further investigated and practical recommendations gathered, the main areas would be secrets management and VCS security based on the survey results.

Competence and knowledge of software supply chain security practices should be improved to prepare for the demand already seen in one customer project. Additional security checks can be easily incorporated into existing build and CI/CD pipelines, in order to help consultants implement these additional checks the framework can help to identify the right tools and provide guidance on the how-to. Some tools and processes could happen earlier in the lifecycle and those could be “shifted-left” to happen earlier and provide faster feedback.

After analysing the gathered data, identified strengths and weaknesses and other findings from the conversations with the case company, the following five elements were identified as specific focus areas for a literature review to discover applicable existing knowledge:

1. DevSecOps
2. Software Security Lifecycle Phases
3. Software Supply Chain Security
4. Secrets Management
5. Securing Source Code Management.

Summarising the focus areas for this thesis: First, evolving from DevOps to *DevSecOps* culture by extending continuous delivery practices to security. Second, practical security practices to take during the different *Software Security Lifecycle Phases*. Third, *Software Supply Chain Security*. Fourth, *Secrets Management*. Fifth, *Securing Source Code Management* systems.

Next, in section 4 these focus areas will be discussed to prepare for Section 5, proposal building.

4 Existing Knowledge on Securing Software Modern Software Delivery

This section of the Thesis discusses the existing knowledge on how to implement and integrate security into modern DevOps driven software development process. It's important to understand that the case company focuses their consulting services around improving the DevOps and Continuous Delivery practices of customers, thus it's essential for the thesis to find out how security related processes can be integrated into those practices and workflows. In addition to focusing on integrating security into these highly automated processes, it's also important to find out what other fundamental practices are important for software development and delivery teams in the modern day.

4.1 DevSecOps

In order to understand DevSecOps, it is first important to understand the original DevOps movement and what are the core practices that go into a DevOps strategy and make it highly effective for delivering reliable software at a fast pace. The next sub-section introduces these concepts. After introduction to DevOps, the DevSecOps term and it's significancy will be discussed in a dedicated sub-section.

4.1.1 DevOps Driven Software Delivery Process

Modern day software development and delivery processes utilise a DevOps approach to delivery software at high speed and volume. Mature organisations practicing DevOps are known to deploy new versions/releases of their software services multiple times per day.

DevOps is a way of thinking and a way of working, it's a framework for enabling teams to practice their crafts in effective and lasting ways. (Davis et al. 2016, 3.) A world where product owners, development, QA, IT operations and infosec work together towards a common goal and are always working on ways to reduce friction for the team. Creating systems that enable developers to be more productive and get better outcomes, by adding the expertise of QA, IT operations and infosec into delivery teams, and automated self-service tools and platforms, teams can use that expertise without being dependent on other teams. These are the outcomes that result from DevOps. (Kim et al. 2021, 2.)

Many think of DevOps as specific tools, but tools alone are not DevOps. What makes a tool DevOps friendly is the manner of its use, not the tool itself. (Davis et al. 2016, 3.)

The foundation of DevOps come from Lean, the Theory of Constraints, and the Toyota Kata Movement, but many also view DevOps as logical continuation of the Agile movement that began in 2001. (Kim et al. 2021, 15.) DevOps borrows techniques such as Value Stream Mapping and Kanban boards from Lean, which were techniques codified for the Toyota Production System in the 1980s. These techniques are all focused on how to create an efficient and effective flow of value to the customer. One of the key principles in the Agile Manifesto (Agilemanifesto.org 2001), which was signed by the leading thinkers in software development in 2001, was to deliver working software frequently and emphasized small batch size and incremental releases instead of waterfall releases. (Kim et al. 2021, 16-17.) One of the key metrics that teams practicing DevOps should pay attention to is the deployment lead time, which is the time between a code being checked into version control and that change successfully running in production. Focusing on having the testing and operations happening simultaneously to the design/development, enables fast flow and high quality when working in small batches instead of large batches that are designed/developed ahead of time before testing and operations. (Kim et al. 2021, 19.)

The 2017 State of DevOps Report shows that high performers deploy code 46 times more often that low performers, which haven't adopted DevOps effectively. The State of DevOps Reports and the research behind clearly shows that when measured, the teams practicing DevOps deliver software faster and with higher quality/less defects. (Forsgren et al. 2018, 42-43.)

The main technical practices that teams should be practicing when practicing a DevOps strategy is Continuous Integration and further, Continuous Delivery which takes Continuous Integration to its logical conclusion, integrated and deployable software. (Vehen 2018, 2-3; Farley et al. 2010, 4.) Next, these important practices will be introduced in more detail.

4.1.2 Continuous Integration and Continuous Delivery

Continuous Integration was first introduced in Kent Beck's book Extreme Programming Explained which was published in 1999. The core idea of Continuous Integration is that

when multiple people work on the same code base, they should integrate their work as often as possible. And the innovative solution to this was to use a centralised build server that merges code from multiple developers and then runs tests to make sure the software still builds and runs. If the automated Continuous Integration build and test process fails, the development team should immediately fix the build before moving on with the development tasks, which means Continuous Integration isn't only a technical practice or process that can be put in place without consensus amongst the development team. (Farley et al. 2010, 55-57.)

In 2010, David Farley and Jez Humble published a book called Continuous Delivery, which popularised the term Continuous Delivery. Continuous Delivery at its core means that software should be always releasable. Which is achieved through practically extending the Continuous Integration practice to test each change to the code base well enough so that it could be released to the users. In order to practically achieve this goal, the book describes a concept called the deployment pipeline. The Continuous Delivery book describes deployment pipeline like this: *"The deployment pipeline has its foundations in the process of continuous integration and is in essence the principle of continuous integration taken to its logical conclusion."* (Farley et al. 2010, 4.)

Figure 2 below shows an example Deployment Pipeline:

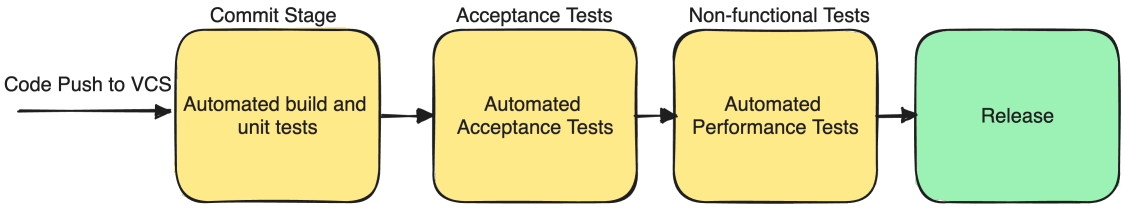

Figure 2. Example Deployment Pipeline, re-drawn. (Continuous Delivery, p 3-4, 106).

As seen in the left side of Figure 2, the deployment pipeline is triggered by code pushed into the Version Control System (VCS), pushing the code creates a new version of the software which is typically referred as a release candidate.

Ideally, the deployment pipeline is a fully automated and run for every change to the code base. This means that every change becomes a release candidate and if it passes the tests in the pipeline, it can be released. Ideally, these changes are small, and the deployment pipeline is run multiple times a day. Although not shown in Figure 2, the pipeline can, and should, run multiple tests/steps in parallel whenever possible since the faster the pipeline reaches a failing test, the faster the developer gets the feedback and can implement a fix. (Continuousdelivery.com 2010)

Same as with Continuous Integration the people working on the software product are all responsible to fix the source code if the pipeline fails (Farley et al. 2010, 3-4). As Continuous Delivery needs to test the software in a production-like environment, the operations team(s) need to be involved in the process of building the pipeline or it will never prove the release readiness of the release candidate. In Continuous Delivery everyone is responsible for the delivery process and people with different roles inside the organisation should all collaborate into the deployment pipeline tests to assure the release candidate can be released if it passes the tests. (Farley et al. 2010, 28.)

In this thesis, the terms Continuous Integration and Continuous Delivery can also be expressed as acronyms CI and CD, respectively, and the Deployment Pipeline is also called Delivery Pipeline interchangeably, since that’s also common in literature.

4.1.3 DevSecOps

The “DevOps movement” has transformed how software is developed, built, and delivered. DevOps makes developers responsible for also operating the software, famously dubbed “you build it, you run it” by AWS CTO Werner Vogels (Gray et al. 2006; Podjarny 2021). In many cases, security is however not truly the responsibility of the development team, even in organisations that boast DevOps culture. Typically, security is owned by a separate, often centralised, teams which causes a bottleneck. Developers don’t get timely feedback when their work is evaluated by the security team after the development is done, or worse, the software is already released. Also, this causes a gap where the security team lacks information of the application, and the developers lack information about security. (Podjarny 2021). To close the gap, the development team must take additional ownership and closely collaborate with the security team, this is a shift from DevOps to DevSecOps culture. However, this does not mean there should be a DevSecOps team, instead the teams just collaborate with the same tooling and in the same delivery pipelines to “bake in security” into the application instead of trying to add it on top. (Podjarny 2021; Bird 2016.)

4.1.4 Shift Left Security

Organisations should strive to “Shift Left” with security in order to “bake in” security, what this means in practice is that security tests and checks are done earlier (as early as possible) in the delivery pipeline. (Bird 2016) (Forsgren et al. 2018, 81) Making this shift permanently means that an organisation must move from DevOps to a DevSecOps culture within the development, operations, and security teams. This means teams should come together and include continuous security and test-driven security practices in the delivery pipelines. (Vehen 2018, 8-9) Developers should take responsibility of security when it comes to what they build.

> “A security strategy that isn’t owned by the engineering teams won’t survive for long and will slowly degrade over time. It’s critical for the security team to define, implement, and test, but it’s equally critical to delegate ownership of key components to the right people.” (Vehen 2018, 12)

According to the Accelerate research on the data between years 2014-2016 of the State of DevOps Reports, high performing teams that include security personnel and security

checks in the delivery process does not slow down the development process, but instead it can even improve the delivery performance of the teams. (Forsgen et al. 2018, 80,92.)

The team already practising Continuous Delivery should have no problems adopting additional, automated, security tests in the delivery pipelines. Security can be seen as just an additional non-functional requirement testing.

4.1.5 Test Driven Security

Many security vulnerabilities come down to simple misconfigurations due to lack of knowledge or simply typos. By creating tests, a security expert can share their expertise to development teams in an executable format, when executed, the test can then statically analyse the code or the running application for security-related threats and misconfigurations. Manual testing in DevOps is considered the exception and automated testing the norm, thus these tests should be automated and continuously executed in the pipelines. It's worth noting that test driven security does not eliminate the need for continuous monitoring of production environments, but it does significantly reduce the risk of a security breach. (Vehen 2018, 10-12.)

In this section the best practices discussed on how to integrate security practices into modern development and delivery processes are core knowledge for the conceptual framework of this thesis. The foundational DevOps strategy and the main technical techniques and practices, Continuous Integration and Continuous Delivery should include security in the process as well. In addition to the techniques and tooling, shifting security left and fostering DevSecOps culture is a key element to foster a successful organisational change in security practices.

4.2 Software Security Lifecycle Phases

Cloud Native Computing Foundation (CNCF) published "Cloud Native Security Whitepaper" in 2020, and the second revision of the paper in 2022. (CNCF 2022) This paper contains a very practical and detailed model of what security practices to implement for "Cloud Native" application delivery. Before discussing the contents, it's important to understand what CNCF means by "Cloud Native", below is the CNCF's definition of "Cloud Native":

”Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.” (CNCF 2018)

As seen by the definition, cloud native applications are a modern take on how applications should be built and what kind of supporting technologies are common. This has implications especially for the tooling used for securing the development and delivery of these applications, next we will discuss some of the key ideas in the paper that are relevant for this thesis.

The CNCF Cloud Native Security Whitepaper splits the Cloud native application development into four lifecycle phases: (1) Develop, (2) Distribute, (3) Deploy, and (4) Runtime (CNCF 2022). This split makes it easier to grasp and makes it concrete what best practices to apply at each phase and what kind of tools can be used. The lifecycle phases can also be useful to map to different roles in an organisation, making responsibilities clearer when it comes to security. Figure 3 below shows a visualisation of the four lifecycle phases and how the workflow follows through the pipeline:

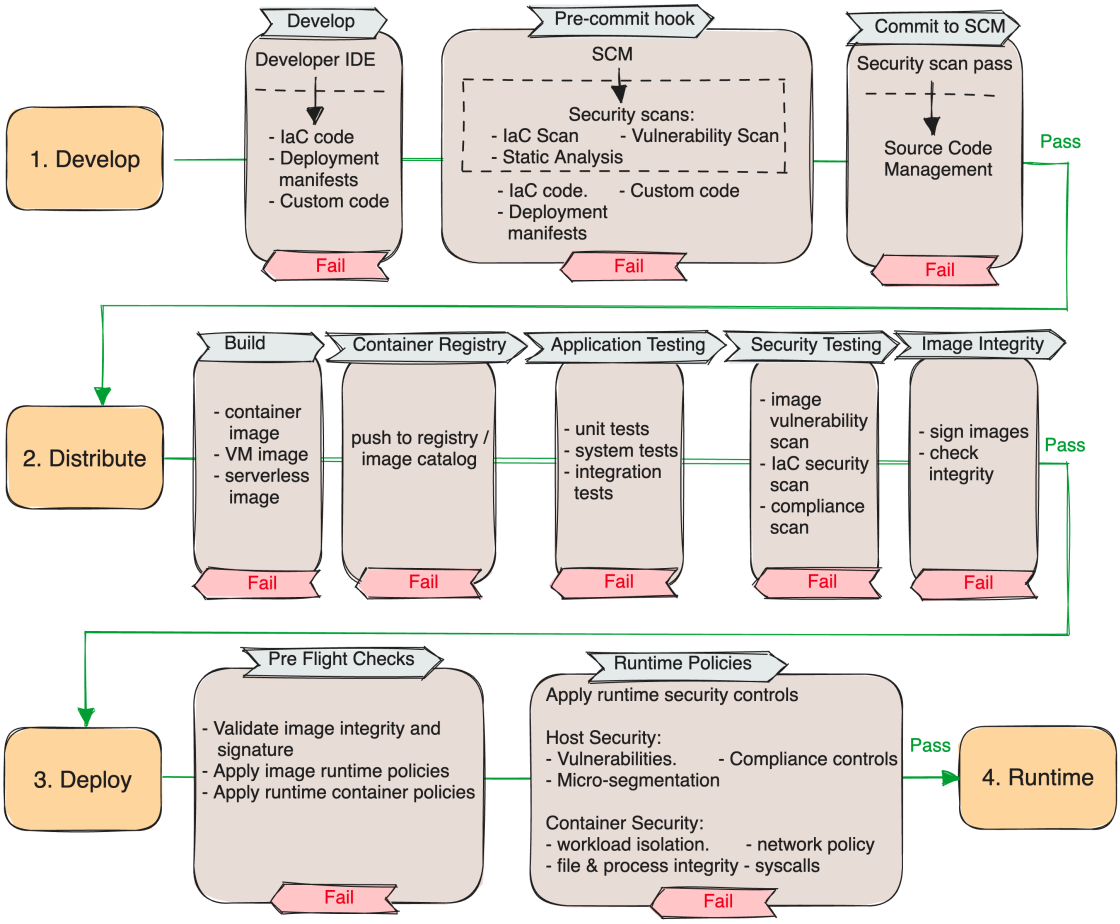

Figure 3. Re-drawn diagram of the Lifecycle Phases (CNCF 2022, 10, 12, 18).

As seen in Figure 3, the first three lifecycle phases form a pipeline that starts with the develop lifecycle phase and ends at the runtime lifecycle phase. This journey is called *a lifecycle pipeline* in the paper. This pipeline is effectively a Continuous Delivery deployment pipeline with a release candidate progressing through it. It's also noteworthy that not only the custom source code is checked for security, but also the supporting code, such as IaC code, this is important because in a cloud native environments a lot of the infrastructure concerns can be declared in code before deployment, allowing effective security and compliance check to be done prior to deployment itself. (CNCF 2022, 5-6.)

Next is a brief description of each lifecycle phase and what activities and best practices should be applied at each phase.

4.2.1 Develop

In the development phase developers write code and configuration such as application source code, Infrastructure as Code (IaC) and container manifests. These artifacts can be analysed by automated tools for known misconfigurations or internal policy violations. These tools can be used for security checks that can be run as early as in the Integrated Development Environment (IDE) of the developer, or in the commit stage of the CI pipeline. One of the main goals for these checks is to provide fast feedback to the developer who can then ideally deal with the security threat immediately before it goes any further in the pipeline or into production. These tests can typically be run in parallel in an already in-place CI pipeline, thus not slowing down the development cycle. (CNCF 2022, 5-6.)

Misconfigurations are the biggest reason for cybersecurity incidents in the cloud according the Check Point's 2023 Cloud Security Report (Check Point 2023). Teams practicing best practices, such as, Infrastructure as Code (IaC) can utilise security scanners to gain feedback on obvious misconfigurations within seconds. These misconfigurations are incredibly common (Truffle Security 2022; Sheridan 2020) and in most cases a security scanner would be able to catch the most obvious issues, such as publicly open S3 buckets, if only they were integrated into the development flow and the changes are made using IaC tooling.

In addition to automated tests, the develop phase can also contain additional code review. This can be done in a pull request in the VCS, for example. (CNCF 2022, 11-12)

4.2.2 Distribute

After the development of new code or configuration, the software should be built and the artifacts distributed to a central registry. Teams practising continuous delivery have sophisticated CI/CD pipelines in use and run automated tests to holistically test the software's releasability. At this stage often additional third-party dependencies, such as open source components are introduced when the software is built. It's important to conduct security scans on all third-party dependencies. Often open-source dependencies can be checked for known vulnerabilities by tools automatically, including such step in the CI/CD pipelines is low effort and huge gain for the overall security

posture. In addition to the security checks done during the develop phase, it's possible to build the software now for dynamic security tests. (CNCF 2022, 12-17.)

At the distribute lifecycle phase, many software supply chain practices are also important, those are described below in the Software Supply Chain Security section, this includes practices such as signing artifacts and generating meta data about the build process and environment.

4.2.3 Deploy

The deploy lifecycle phase can be thought of as "pre-flight" checks. It's adamant to validate the application configuration and the integrity of the artifacts before a deployment to a production environment is carried on. It's also possible to include tests for compliance, such as verifying tagging or labelling of the deployment at runtime. (CNCF 2022, 18.)

4.2.4 Runtime

Runtime lifecycle phase, also referred to as "Runtime Environment" in the paper, differs from the first three lifecycle phases because at this point the deployment pipeline is done, and the application is running in the production environment. The runtime lifecycle phase is empowered by the earlier phases, most issues should already be caught at this point but some threats, such as previously unknown vulnerabilities, might still be able to be exploited and thus there must be continuous work to monitor and analyse the runtime environment. Runtime phase consists of three critical areas: (1) Compute, (2) Access, and (3) Storage. (CNCF 2022, 19.)

A modern Cloud-Native infrastructure typically consists of many layers in which all these three areas present themselves, the application itself runs in the top layer. Figure 4 below shows a typical layered runtime architecture.

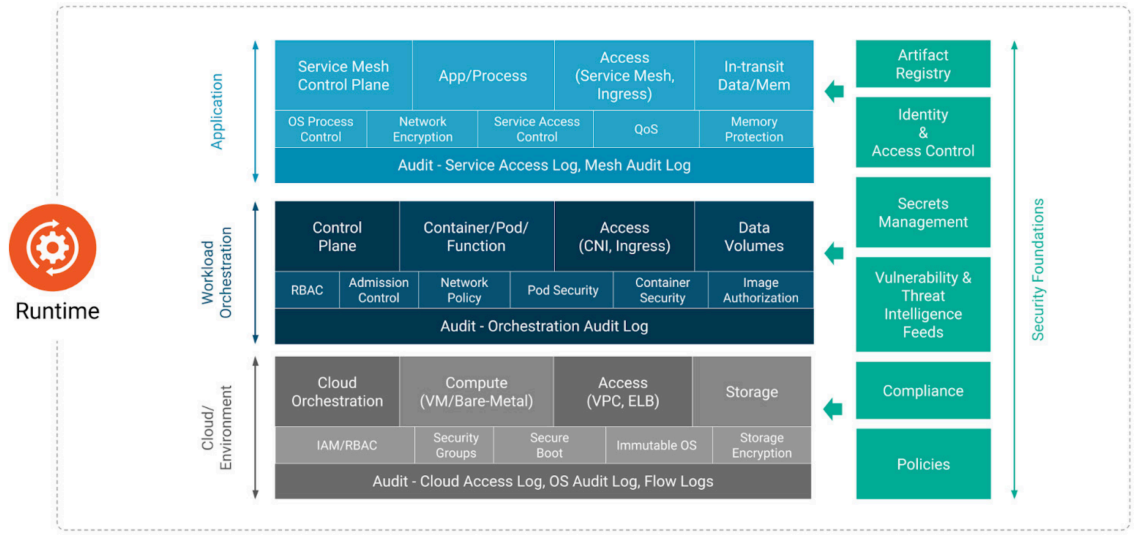

Figure 4. Runtime Lifecycle Phase (CNCF 2022, 19).

As seen in Figure 4, the runtime lifecycle phase is very complex and fundamentally different compared to the lifecycle phases prior to the actual deployment of the application. Main activity during runtime is monitoring and auditing, and there are many things to monitor and many logs to audit. As seen in Figure X, there commonly are three layers: (1) Application, (2) Workload Orchestration, and (3) Cloud/Environment. (CNCF 2022, 19.) In all these layers are common building blocks.

In addition to the three critical areas compute, access, and storage, there are concerns around networking and large amount of so called “Security Foundations” that support the security posture of all the layers. The most relevant of these foundational security practices, for the case company, are discussed next in this section of the thesis.

4.3 Software Supply Chain Security

A supply chain is a process of getting a product to the customer. In software development and delivery, there is also a supply chain which consists of producers and consumers of software components. (CNCF 2021) More technically, software supply chain is the sequence of steps a producer takes that result in the creation of a software artifact for a consumer. (SLSA.dev 2023).

According to the CNCF Software Supply Chain Security Whitepaper (CNCF 2021, 8), the main difference between a traditional supply chain of physical products and the software supply chain is that the software supply chain is:

- Intangible – software is intangible, made up of virtual and digital components which makes it difficult to count, discover and understand end-to-end

- Mercurial – software changes faster than the physical work, the supply chains are under constant and rapid change

- Iterative reuse – one supply chain depends on other supply chains, that again rely on other supply chains (of downstream software components) (CNCF 2021, 8-9.)

Software supply chain security affects almost all software being develop today as almost all software products include open-source dependencies. According to the Synopsys Open Source Security and Risk Analysis report 96% of codebases analyzed in the report contained open source dependencies. (Synopsys 2023.)

These open source and even commercial dependencies possess a risk for the application that depends on them as they might change unexpectedly or become unmaintained and never receive important security patches. Keeping track of these dependencies and the transitive dependencies is becoming increasingly important as the threat actors are increasingly targeting the supply chain in their attacks, as seen in Figure 5 below.

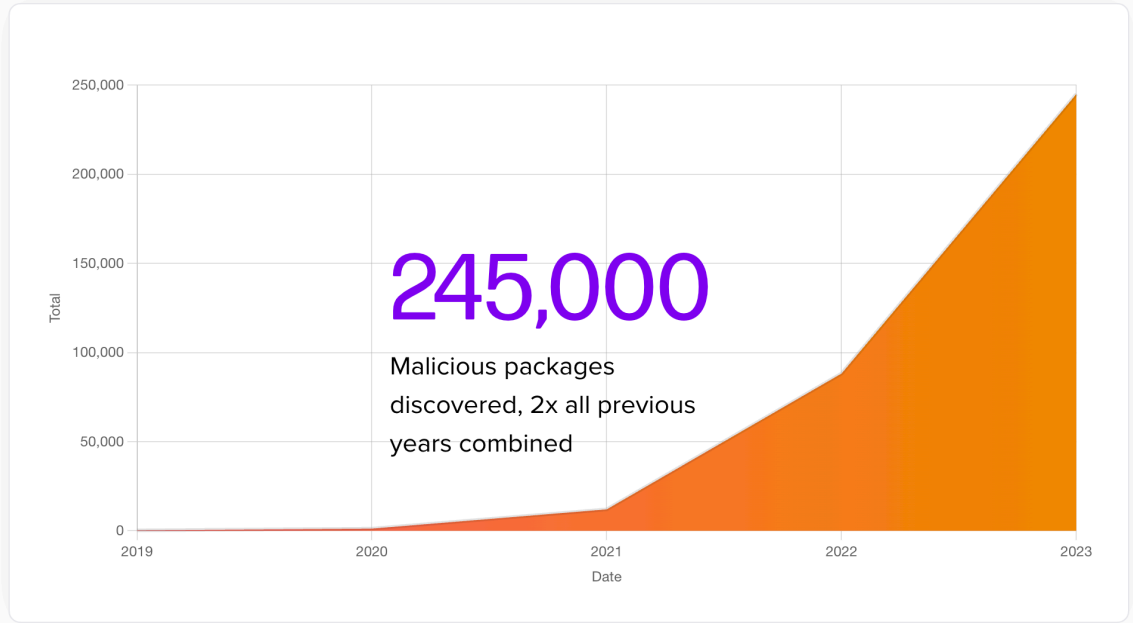

Figure 5. Diagram of the amount of observed Software Supply Chain attacks between 2019-2023 (Sonatype 2023)

Figure 5 above shows how the malicious packages discovered in the Sonatype research is increasing rapidly, clearly indicating that the software supply chain is being targeted by cyberattacks more and more every year.

Securing the software supply chain against threat is a very extensive problem, as illustrated by Figure 6 below.

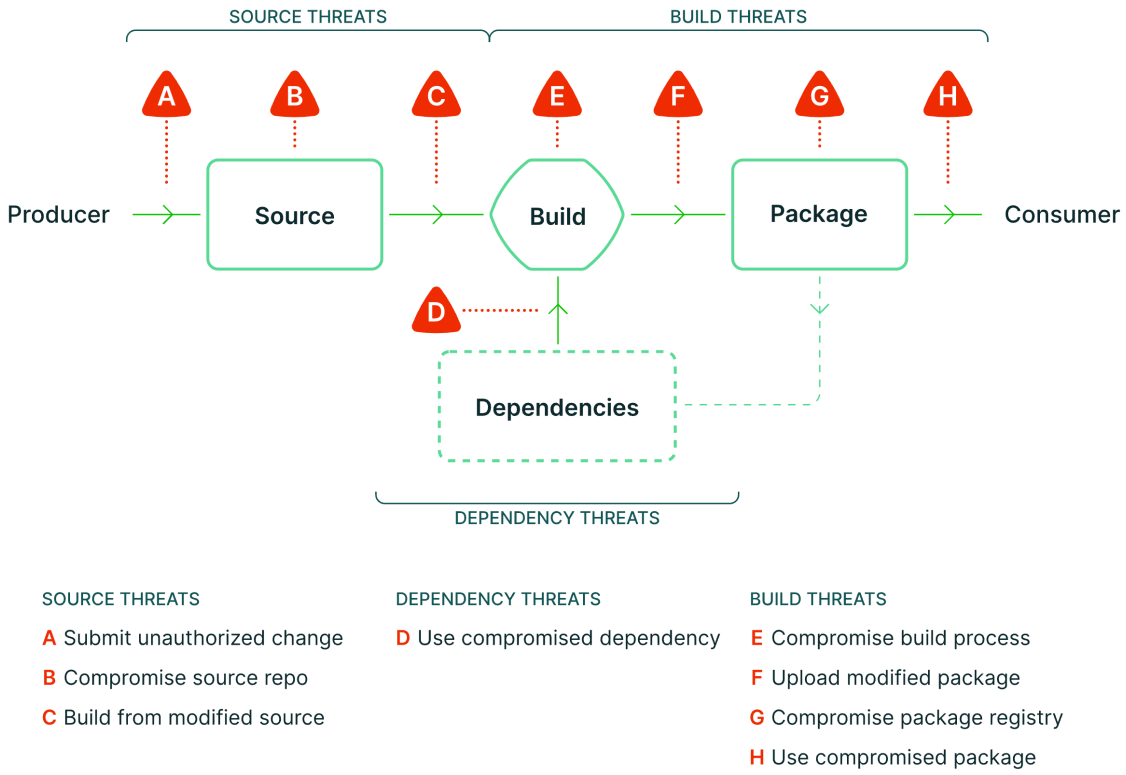

Figure 6. Supply Chain Threats. (SLSA.dev 2023).

Figure 5 shows a worrying picture of the many threats that a software supply chain faces. Looking at the figure closely, it's clear that the software could be compromised and tampered with at any point of the supply chain if none of these threats are addressed. The SLSA website also gives a real-world example of each of the threats being exploited, so the threat is not just theoretical. (SLSA.dev 2023.)

According to the CNCF Software Supply Chain Security Whitepaper (CNCF 2021, 8), there are major security threats to organizations due to lack of visibility into consumed software products and implicit trust placed on producers. Instead of trusting producers there's a need to verify the trustworthiness and integrity of software products when an organization is ingesting software dependencies or products. (CNCF 2021, 3-4.) To mitigate these security threats and to improve the security posture of the software supply chain, next we will discuss three important technical practices: signing, provenance and SBOMs.

4.3.1 Signing

All software artifacts built within an organization should be digitally signed in a way that the signature can be cryptographically verified against the party that is expected to have signed the artifacts. (CNCF 2021, 5) According to the CNCF (2021), Software Supply Chain Security Whitepaper signing and verification should happen at every step:

> ”The signing of artefacts should be performed at each stage of its life cycle, along with the verification of signatures from prior stages, to ensure end-to-end trust.” (CNCF 2021, 34)

As mentioned in the above quote, the signatures provided by the producer of an artifact should be verified by the consumer or otherwise there’s no improvement to the security posture.

Signing is not limited to the built software that is often thought as *the artifact.* In addition to signing the software artifacts, in high assurance environments additional information / metadata about how the artifact is produced should be signed as well, below are some examples of what can be signed in addition to the software package:

- Commits are signed by developers before pushing to the Version Control System (CNCF 2021, 34-35)

- Inputs, outputs and logs of a build step are collected and signed (CNCF 2021, 11-12)

- Signing configuration files (CNFC 2022, 17).

As seen from the above, signing can be quite a complex problem to solve for each organization. But luckily there are excellent open-source projects such as sigstore, in-toto framework and Tekton Chains that aim to make these tasks easier for everyone in the industry. (Linux Foundation 2023; Linux Foundation 2023a; Linux Foundation 2023b.)

4.3.2 SLSA

SLSA is a set of incrementally adoptable guidelines for supply chain security, established by industry consensus. SLSA consists of different tracks and incremental levels of maturity allowing organizations to gradually adopt the framework. However, current SLSA v1.0 specification only addresses build threats that were shown in the figure 5. In future SLSA will likely expand to include advice against additional threats such as source and build infrastructure. (SLSA.dev 2023a.)

For the SLSA build track the main new activity a software producer needs to adopt is creation of provenance attestation from the build pipelines. Provenance is metadata from a build pipeline, and it is a type of attestation. But what is an attestation in the software context? In the real world an attestation can be your driver's license, it is proof that you are allowed to drive a car (and/or other vehicles). But a software attestation according to SLSA website is:

> "A software attestation is an authenticated statement (metadata) about a software artifact or collection of software artifacts." (SLSA.dev 2023b).

SLSA framework includes a provenance format which is in practice just fields that need to be filled with metadata gathered during the build, but in a very specific way to ensure the integrity and trustworthiness of the provenane. Publishing a provenance document along with the software artifact(s) allows the consumer to verify how the artifact was produced. For example, SLSA provenance can help consumers answer these questions:

- Which exact version of the source code is in this software artifact?

- From which source code repository does it software artifact come from?

In order to trust the provenance document information, the provenance document must also be digitally signed in a way that the consumer can verify that the identity of the signer matches with the consumer's expectation. Verifying this trust chain today is challenging because tools don't often support such verification steps out-of-the-box, and the standards are still being actively developed and unstable. However there are some notable tools that can help, for example with signing the Linux Foundation's sigstore project provides free-to-use public infrastructure which is easy to use. (Linux Foundation

2023) And recently the Node Package Manager (npm) tool started supporting both generating and verifying SLSA provenance as a built-in feature in the core command-line tool. (DeHamer et al. 2023.)

4.3.3 SBOM

Software Bill of Materials was deemed important and was championed in the US by the National Telecommunications and Information Administration (NTIA) starting from 2018. (NTIA 2018) However, it wasn't until the software supply chain was attacked in cases such as the SolarWinds SUNSPOT malware (Crowdstrike.com 2021) that affected many organizations and governments across the globe. Quickly after, in 2021 the US and EU legislation finally started to demand organizations to provide SBOM documents if they want to sell software to the government. (White House 2021; White House 2021a; European Commission 2022.)

Software Bill of Material lists all the dependencies (typically libraries) that are bundled into an artifact. Figure 7 below visualizes this mapping.

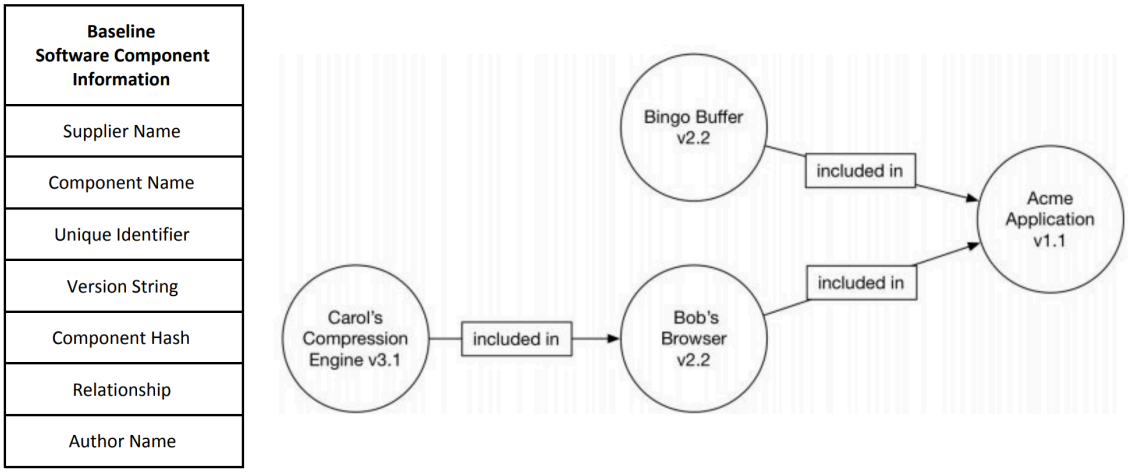

Figure 7. Software Bill of Materials visualized along with baseline software component information (Figure 1, NTIA 2020).

Figure 7 also includes the NTIA defined Baseline Software Component Information which is the bare minimum that is needed as metadata of each component in the SBOM. Unfortunately, there is still multiple competing data formats for SBOMs, mainly SPDX and CycloneDX, but both are very actively used and being developed. For producers

and consumers a single standard format would be more practical, but hopefully in future one format becomes the standard approach. (NTIA 2020.)

These SBOM documents can be used for many things, such as scanning the list of “materials” for known vulnerabilities, producing release notes or analyzing the impact of a newly found vulnerability by querying a catalog of SBOMs for all the products within an organization. Especially for identifying known vulnerabilities the SBOMs are very relevant today since the Sonatype’s State of the Software Supply Chain report shows that over 96% of the downloaded vulnerable packages from Maven Central ecosystem had versions available with the vulnerabilities patched, meaning that in 96% of the cases using a known-vulnerable package could have been avoided. (Sonatype 2023.)

There are many open source tools that can help a producer to generate an SBOM such as Syft, and there are also projects aiming to help with keeping a catalog of SBOMs for queries such as Guac. (Anchore 2023; Guac.sh 2023.)

Software supply chain security is still a fast-evolving space but it has quickly become an interesting and important area that affects everyone working with software development or delivery, thanks to it being relevant on both the producers and consumers of software artifacts. The foundational nature of the security threat imposed by poor software supply chain practices makes it an important part of the foundational security practices of the conceptual framework of this thesis.

4.4 Secrets Management

Users of software have had secrets management techniques for as long as there has been secrets, these systems range from really good (password managers, physical safes) to extremely bad (post-it note under the keyboard). (Dotson 2019, 36.) Secrets management provides assurance that resources, platforms, and cloud environments can only be accessed by authenticated and authorized entities, not only humans but also applications, automation tools and scripts. (Cyberark 2023) Secrets management is thus somewhat up to the user to figure out, or a dedicated system might be used (such as a password manager software). Also not only are human users concerned when it comes to secrets management but also the way secrets are provided to or stored in applications and in automated scripts.

Secrets are used by humans and applications when building and operating software applications and infrastructure. Secrets are anything that is presented by a human or application to access a system. For example, secret can be a password or an API token. Securing applications is especially tricky because they cannot complete a challenge such as Multi-Factor Authentication which humans can do on their phone as an example. When an application needs to authenticate, for example, when connecting to a database, there needs to be either the secrets for the authentication already present on the server for the application, or the application can rely on an external service to fetch those secrets when needed. However, this still has an obvious problem; how does the application authenticate with the external “secrets” service? Such a problem is typically solved by bootstrapping application servers or workloads with some sort of identity, such as a certificate that is signed by a trusted, internal, authority. However, in public cloud platforms such as AWS and GCP, there’s a metadata service available for workloads that can be used by the workloads to discover their identity and some metadata about themselves, which can be used as proof of identity when authenticating. Using these services can save an organisation from a lot of complexity compared to the organisation self-hosting such workload identity infrastructure. (Dotson 2019, 36-40.)

Next, we will discuss a more specific problem faced by many fast moving DevOps driven software teams and organisations, secrets sprawl.

4.4.1 Secrets Sprawl

Application infrastructure and the speed of development has been greatly accelerated by the advances made in technology and by new offerings such as cloud platforms that make it possible to provision infrastructure ad hoc instead of waiting for servers to be delivered to your datacenter. As the development speed has increased, it also means more and more secrets to manage for organisations which it has caused a problem called Secrets Sprawl where secrets are scattered around the organisation on developers’ laptops and sticky notes. This exposes organisations to risks if these secrets are leaked, so secrets management requires special tooling and discipline to help manage these sprawling secrets in a centralised and controlled manner. (Dadgar 2018)

4.4.2 Secrets Managers

To mitigate these problems many organisations use special software for secrets management to keep secrets stored in one secure place instead of spread all over the application landscape. In addition to storing the secrets, the secret management software helps to provide a detailed audit log of who/what accessed and which secrets. When secrets are not centrally stored there's no central source of truth for rotating secrets either and it's hard to share a secret with a colleague securely. (Dadgar 2018)

Many cloud providers have their own secrets manager/management service that is typically easy to use when operating inside the provider's cloud platform. For on premises or multi-cloud use-cases one popular option is to use HashiCorp Vault. (HashiCorp 2023) It's worthy to note that these applications are not meant for humans to store passwords for their personal logins, but for operators to manage secrets for application workloads.

4.5 Securing Source Code

Version control systems, also known as source control systems or source code management tools, are software tools that help software teams manage changes to source code. The version control system software keeps track of every modification to the code in a specialised database. (Atlassian.com 2023) Version Control is a system that records changes to a file or set of files over time. It's commonly used for software source code, but it can be used with nearly any type of file on a computer. (Chacon et al. 2014, 10.) Version control systems are used to track changes to source code, not only are developers using version control systems but also other professionals can use it to track their changes to scripts or whatever text-based files they have to store. When using a Version Control System it's possible to collaborate with others easily with everyone having access to the versioned files from the system. (Chacon et al. 2014 10-13.) Securing access to source code is important for many reasons, like Atlassian states in their website:

> ”For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected.” (Atlassian.com 2023)

Productised Version Control Systems (VCS) such as GitHub and GitLab have quickly become an essential and central place for developers, operators and other stake holders to contribute on source code. Source code is not only limited to application source code thanks to technologies such as IaC which allow operations to develop re-usable infrastructure configuration/templates that are versioned and persisted in a VCS as best practice. VCS systems also are commonly used to trigger build and deployment pipelines, and other kind of automated tasks. Which means it's an extremely valuable target for threat actors. Not only can they tamper with, leak, or encrypt (for ransom) all the application source code, through the deployment pipeline credentials, a threat actor can gain highly privileged access of the cloud infrastructure and private data of the target organisation. (OpenSSF 2023)

Luckily, popular VCS platforms such GitHub and GitLab have realised these threats and have started offering more and more security enhancing features that can be activated to protect source code from tampering, some of these common features are described next in the following sub-sections.

4.5.1 Access Control

The most basic level of access control is the visibility (the ability to view or edit) of the version control repository, depending on the system used there can be different controls available, but depending on the sensitivity of the code stored in the repository, only authorised personnel should have access to it. The visibility is extremely important to protect against human mistakes, such as accidently adding secrets into the repository, if the repository is only visible to few people the incident is less severe than if the repository is visible for anyone with the link to it. (GitLab 2023.)

GitGuardian (GitGuardian 2023) is a code security platform that kindly scans public GitHub repositories for secrets and notifies the users that have committed them into the repository to remove the leaked secrets (GitGuardian 2023a). However, having a tightly controlled visibility does not mean that secrets should end up in version control system, this is considered a bad practice. The section on Secrets Management described these problems and potential solutions in more detail. (GitGuardian 2023a.)

Some of the other basic features that should be enabled from a security perspective are multifactor authentication for users and branch protection rules. Multifactor

authentication is crucial because passwords can be leaked and it can be easily enforced at organisation, for example in GitHub. (Vehen 2018, 153.)

Branch protection can protect the code base against unauthorised changes for example, you can enable a rule requiring a peer developer to approve any code change before it is merged to the main branch, making it harder to tamper with the source code even if a developer's credentials were leaked. (GitHub 2023.)

4.5.2 Monitoring

In addition to the security controls described above around access control if a user is compromised those settings can potentially be altered by impersonating the user of the compromised credentials themselves. Thus, it's important to monitor the events and the configuration of different version control repositories of an organisation over time.

OpenSSF Allstar (OpenSFF 2023a) is a project built specifically for GitHub monitoring to make sure the configuration adheres to security best practices. The tool can be used to define the policies that organisation must adhere to and then continuously monitor that the configuration matches the policies. If a policy is violated, Allstar can either change the configuration setting automatically or alert on it. (OpenSFF 2023a.)

It's also important to monitor for changes needed to user permissions, for example removing users that have left the organisation. Automating this might require writing custom scripts, depending on where user accounts are stored. (Vehen 2018, 153.)

Beyond than what can be included in this thesis. In 2023, OpenSource Software Security Foundation (OpenSSF) released Source Control Management (SCM) Best Practices Guide which is an extensive set of best practices for GitHub and GitLab SCM platforms which are the industry giants. These best practices cover areas such as:

- Hardening CI/CD pipelines against supply chain attacks
- Branch protection policies for healthy coding workflows
- Recommended access controls and permissions

- Server-level policies for globally enforced best practices. (OpenSSF 2023b)

In addition to the set of best security practices, the guide recommends tools in addition to the already mentioned Allstar (OpenSSF 2023a) that can be used to continuously monitor and audit that all the source code repositories under the organisation use the security best practices (OpenSSF 2023a). These tools are good examples of the continuous security mindset, it's not enough to document and recommend the security best practices, instead they should be continuously monitored and audited to identify risky configurations or potential security incidents.

As mentioned at the beginning of this section, securing the version control system is important to protect the organisations "crown jewels". When DevOps and continuous delivery practices are tightly integrated into with the version control systems, securing the access and effective use of the available security controls of the given tool is are crucial best practices for this thesis and one of the core foundational security elements of the conceptual framework.

4.6 Conceptual Framework

Figure 8 below shows the conceptual framework of this thesis. The conceptual framework is split into two major parts: people and processes, and implementation. This is to highlight the fact that the literature shows that it's not enough to only take in use and implement new tools, but the whole organisation needs to undergo a change to collaborate across team borders when it comes it security.

<table>
<tr><td colspan="5">DevSecOps Culture (Bird 2016) and
Methods (Shift-left Security; Test-Driven Security) (Vehen 2018)</td></tr>
<tr><td>Secure Software Development Lifecycle Phases (CNCF 2022; Farley et al. 2010)</td><td>Develop & Integrate</td><td>Test & Distribute</td><td>Deploy</td><td>Monitor</td></tr>
<tr><td>1. What is the activity?</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2. What information and artifacts are available?</td><td></td><td></td><td></td><td></td></tr>
<tr><td>3. What could be checked for? & Most important?</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4. How to check/achieve it? (tool/practice)</td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="3">Foundational Security Practices
(CNCF, 2022)</td><td colspan="4">Software Supply Chain Security (SLSA.dev 2023; CNCF 2021)</td></tr>
<tr><td colspan="4">Secrets Management (Dotson 2019)</td></tr>
<tr><td colspan="4">Version Control System Security (OpenSSF 2023)</td></tr>
</table>

Figure 8. Conceptual Framework for a security framework.

Starting with *the People and processes* in Figure 8 above, it's further split into two parts: culture and methods. First, Figure 8 shows that a DevSecOps culture is an important consideration for improving security posture of for an organization. It's important to note that this move will be harder for organisations that don't have a strong existing DevOps culture. Second, literature shows that high performing teams include security personnel early in the development process, thus shifting security left is an important method. This is where the Test-Driven Security method comes in. According to (Farley et al. 2010, 13-14.), security checks should be automated and added as part of the deployment pipeline. This will give the developers quick feedback and allows them to act on the feedback without waiting for the security personnel to check their work after it's fully completed.

In addition to people and processes, the second part of the conceptual framework is *the Implementation.* First, there's the core implementation which builds security practices on top of (ideally) existing Continuous Delivery culture and deployment pipeline. Software Security Lifecycle Phases refers to the lifecycle phases defined in the CNCF Cloud Native Security Whitepaper. This practical guidance is the framework for the core practices and steps to add into the deployment pipeline. Second, there are many foundational security practices that could be included in the foundational part of the

implementation, but guided by the current state analysis three core practices were deemed most relevant and interesting to tackle first: secrets management, software supply chain security and VCS security. For all three of these important categories there are clear best practices and steps that can be taken to improve the security posture of an organization and protect it against threat actors.

This concludes the existing knowledge section of this thesis. Next, in Section 5 the proposal building and the initial proposal will be discussed.

5 Building Proposal for Continuous Security Framework for the Company

This section is about building the initial proposal of the continuous security framework for the case company. The proposal is informed by the findings from the current state analysis (Data 1), by the relevant elements of existing knowledge of the conceptual framework, and by the input from the stakeholders (Data 2).

5.1 Overview of the Proposal Building Stage

This section describes the steps in the proposal building. The aim of the proposal is to propose the security framework that would address the weaknesses identified during the current state analysis and provide actionable initial plan for implementation on improvements on key focus areas. Also, the best practices identified in the key focus areas of the conceptual framework are transformed into practical, tailor-made, suggestions so that they best serve the needs of the case company as the security framework.

The outcome of the proposal building is a continuous security framework. The initial proposal was built in three steps. First, initial visualisations and scoping was done in small workshops to co-create the shape and scope of the continuous security framework. This was a pre-requisite for starting with practical work to dive deeper into the topics and produce the practical information on each element of the framework.

Second, the initial materials were presented as a draft of the initial continuous security framework and feedback was gathered thoroughly based on the comments. During the second step, there were also informal one-to-one interviews on the various elements with different experts of the case company, that helped to deepen the focus and better scope the framework and problems it can solve. The elements and the scope of the framework needed to be clearly defined before the implementation of any practical work can be started. Otherwise, there’s a risk the implementation won’t be used and delivers no value. To achieve a clear and suitable initial framework, many early ideas were discussed in workshops with the case company consultants, and later initial drafts were co-created and discussed further to distil the early ideas and help achieve a reasonable scope for the first iteration.

Third, the draft of the continuous security framework design was updated and then the information was merged from virtual canvases and notes into a website that would be the live up-to-date version of the framework going forward. After this step, the framework would only live in this single place, backed with version control, and it would be easy for others to collaborate with ideas following a well-known process from daily work.

Thus, building the initial proposal in these three steps unsured that everyone is heard, and every idea can be easily incorporated into the framework during first two steps, and it's easy to change in the beginning. In step three, the website was created to make sure all information and future collaboration happens in a single controlled environment, although it's a bit harder to change drastically. Input from stakeholders was taken not only in workshops but also at individual level, to make sure everyone would speak their mind and the inputs were very influential to scoping and sharpening the focus of the framework. It was important to focus on topics that the case company stakeholders deem necessary and important, instead of blindly following existing knowledge that might be aimed for a different context.

5.2 Findings from Data Collection 2

In the very first initial planning and brainstorming workshops, the stakeholders were provided an early draft in the form of an interactive, collaborative, virtual canvas, the canvas was prefilled with early design which was then modified during the workshops with also field notes taken of the suggestions. In addition to working with a virtual canvas, a single face-to-face workshop was held where key information from the conceptual framework was first shared in the form of a short presentation, followed with discussions and brainstorming. Field notes were taken from the face-to-face workshop session.

The early feedback and interest towards the topic was important, and many of the topics identified during the current state analysis were still found interesting and were discussed among the stakeholders during workshops. Also, elements from the conceptual framework, notably Test-Driven Security were found useful to help drive implementation.

In Table 2, Data 2 findings are presented in a way that pulls together the current state analysis findings, conceptual framework and Data 2 suggestions gathered from co-creation.

Table 2. Key suggestions from stakeholders (Data 2) based on initial draft, in relation to the findings from the CSA (Data 1).

	*Key focus areas from **CSA** (from **Data 1**)*	*Inputs from literature (**CF**)*	*Suggestions from stakeholders for the Proposal, summary (from **Data 2**)*	*Descriptions of their suggestions (in detail)*
1	Incorporating security checks into pipelines / shifting left with security processes.	Securing DevOps (Vehen 2018)	a) The topic is really big, needs to be better scoped.	Security affects all phases of the development lifecycle and remains important also in production, it's an overwhelmingly big topic.
			b) It's not what we're tasked with.	Working on security enhancements is rarely the core responsibility of our consultants.
2	Foundational security practices that should be further investigated and practical recommendations gathered, the main areas would be secrets management and VCS security.	Practical Cloud Security (Dotson 2019) VCS System Security (OpenSSF 2023)	a) Getting people to change their workflow and adopt a new tool (e.g. secrets manager) is the challenge, not setting up the tool.	Setting up a secret management system is peanuts. Rolling out the use of a secret management system for a multinational enterprise with many users of varying levels of boneheadedness is another thing entirely.
3	Competence and knowledge of software supply chain security practices should be improved to prepare for the demand.	Software Supply Chain Security (SLSA.dev 2023) (CNCF 2022) (CNCF 2021)	a) Write a blog post about SLSA.	Many consultants are interested to know more about some of the terms (for example SLSA) that they know are related to supply chain security, but don't have experience with yet.
4	Need for a clear structure for implemetation and for effective customer communication about security.		a) Add a Security Landscape diagram to the security framework	A diagram sketched in workshops was found useful and should be used in the framework as well.

As seen in Table 2 above, first, there were two suggestions from stakeholders on the way security processes could be integrated into existing deployment pipelines and how to shift security left. In the conceptual framework the literature gives good suggestions on how to secure software delivery processes under a DevOps strategy, however it is more from the perspective of a dedicated security personnel. The stakeholders were concerned with regards to the scope of the security framework as a whole and whether it will be practical for consultants that don’t specialize in security.

One of the consultants expressed their worry in the following way:

> "...when work will be done around this, picking up the roadmap will surely be challenging." (Consultant 1)

This feedback was key to make sure the initial scope of the framework is fairly limited in order to accurately match with the context of the case company. To address the concern with regards to whether these security tasks match with the typical line-of-work, it was decided to focus on security practices that consultants find most actionable and easiest to adopt in existing or typical projects.

Second, there was a suggestion regarding the foundational security practices around secrets management and version control system security. The consultant raised the suggestion in following way:

> "Setting up a secrets management system is peanuts. Rolling out the use of a secrets management system for a multinational enterprise with many users of varying levels of bone headedness is another thing entirely." (Consultant 2)

As the quote above mentions, the challenge for secrets management and version control security is not a purely technical one. When addressing these security concerns, the literature best practices show that using a tool is common. Taking the secrets management system as an example, this tool would be setup centrally for multiple teams or even for the whole organization, and its purpose is to store secrets that application workloads can then access from this single place, the secrets management system. Adopting the new system requires changes to all existing deployments and to developers and/or operators' workflows when they need to add new secrets or make other changes in the system.

In the framework, this suggestion was taken into account and in further discussions there were additional ideas how to approach the problem still in daily project work. In a nutshell, this suggestion guided the framework towards making sure when adopting a new tool, it's done using the simplest way possible first and in a way that best fits the existing workflow. Once the solution gains adoption, it's then possible to take into use more advanced and secure ways of utilizing the system. Also using a solution that is already available, instead of spinning up a new system, is preferred in most cases. A good example of a solution that's often available is one that is already provided as a service in the cloud provider used.

Third, there was a suggestion with regards to software supply chain security specifically, although it's part of the foundational practices. Software supply chain security is still evolving quickly and there's a lot of confusion around what should be done and how. Thanks to the legislation mandating certain minimum practices (mainly an Software Bill of Materials), it's impossible for organizations to ignore this area of security. The suggestion was to write a blog post about the SLSA framework to raise awareness both internally and externally as a form of marketing. Although the suggestion is out-of-scope for this thesis, it affected the proposal building to make sure effort is put especially into supply chain security content into the framework. As there is clearly interest among the consultants in this topic and demand from customers on software supply chain security and not a lot of concise, actionable knowledge available.

5.3 Initial Proposal

The initial proposal consists of two practical elements: Secure Software Development Lifecycle Phases and Foundational Security Practices. Supported by a higher level, more of a strategic element; DevSecOps Culture and Methods, which serves as an umbrella of guiding principles for the practical elements.

The Secure Software Development Lifecycle Phases element is a slightly modified version of the lifecycle phases described in the CNCF Cloud Native Security Whitepaper (CNCF 2022), the changes and the rationale behind the changes are described later in this section. The DevSecOps Culture and Methods are largely inspired by the literature on best practices on securing DevOps development and delivery processes. The Foundational Security Practices are informed by the leading projects in the corresponding foundational security area and the best practices found from literature.

For all the best practices identified from existing knowledge, the framework goes further into practical suggestions on how to adopt them and where to focus first, in the context of the case company based on the improvements and weaknesses identified in the current state analysis, and the suggestions gathered during the proposal building. Table 3 below shows the initial proposal for the Continuous Security Framework.

Table 3. Initial Proposal for the Continuous Security Framework.

Culture and Methods	**Shift-left Security**			
	Test-Driven Security			
	DevSecOps culture			
Secure Software Development Lifecycle Phases	**Develop & Integrate**	**Test & Distribute**	**Deploy**	**Monitor**
What is the activity?	Writing code and configuration incl. deployment manifests and IaC.	Building, testing and distributing artifacts: container images, binaries, VM images, other packages.	Production deployment.	Monitoring production infrastructure and workloads.
What information and artifacts are available?	Static text files.	Software that builds and runs.	Deployment configuration, signatures, SBOM, provenance, test results.	Metrics, logs (app, network, OS), Cloud APIs and configuration.
What could we check for?	Known vulnerabilities in dependencies, insecure patterns in code/config.	Compliance, insecure runtime configuration, functionality, integrity.	Production readiness according to policies and compliance.	Suspicious activity, unstable software, over-privileged identities.
What is most important?	Fast feedback cycle, CI pipeline must complete in <5min. Catch known issues fast before developer context switches.	Running software and configuration is tested and scanned in production-like environment. Provide timely feedback in <1h. Produce signed artifacts, SBOMs and provenance from build for distribution.	Verify and enforce integrity (signature, provenance), compliance (tags, labels etc.) and runtime configuration in pre-flight checks.	Alerts and reports.
How to check/achieve it? (tool/practice)	Static Analysis tools: trivy, tfsec, conftest etc. Custom policies for misconfigurations. Peer review for major changes (on top of automated checks).	Production-like test environments and automated tests. SLSA provenance, syft (SBOM), sigstore.	Admission controllers (Kyverno), conftest/OPA, custom checks.	Steampipe, Trivy Operator, Tetragon/Falco.
Foundational Security Practices	**Software Supply Chain Security**			
	Secrets Management			
	Version Control System Security			
Security Landscape				

As seen in Table 3, the framework emphasizes the *Secure Software Development Lifecycle Phases*, and this is indeed the core of the framework. The *Foundational*

Security Practices support the core practices, and the *Culture and Methods* serve as the high-level strategy of guiding principles. Finally, the Security Landscape helps users of the framework to map the practices with an example application development and delivery stack. Next, all four elements of the initial framework proposal will be discussed in detail.

5.3.1 Element 1 of the Initial Proposal: DevSecOps Culture and Methods

In order to have success with a security strategy in a DevOps driven organisation, security cannot come late in the delivery process when all the code is ready for a release, nor cannot it slow down the development teams working in small iterations. Thus, a successful Continuous Security strategy and framework should focus on enabling teams to work in a secure way, instead of disabling teams from working in an insecure way. By focusing on enabling the teams, the security processes support the overall delivery and enhance the quality, and naturally, security of the solution. The main methods of enabling DevOps teams to work in a secure way are *Shift-Left Security* and *Test-Driven Security*.

By shifting security left, the developers will get the feedback they need to secure their code sooner rather than later in the process. Getting the feedback as soon as possible is important to minimize context-switching, meaning that the developer can immediately make the necessary fix to the code or configuration instead of having to come back to code or configuration days or weeks later, when the developer has moved on to other tasks. Not every security related scan or check process can be shifted left, but it's a guiding principle for all stakeholders in the development and delivery process to check the work as soon as possible in the pipeline. For example, a misconfiguration or known insecure patterns in code can be checked already at the commit phase to provide feedback on these issues in minutes, not in hours. For the case company a key enablement for shifting left is to practice the best practice for infrastructure provisioning, Infrastructure as Code (IaC). When writing the infrastructure definition as a configuration file, it can be checked by a static analysis tool in seconds to provide quick feedback. Writing everything possible as code is a key practice to enabling effective shift-left security strategy.

The Test-Driven Security concept is put into practice by adding tests to the deployment pipeline. Figure 9 below is an example of how the example pipeline shown in the Existing Knowledge section figure <TODO> can be enhanced with additional security tests.

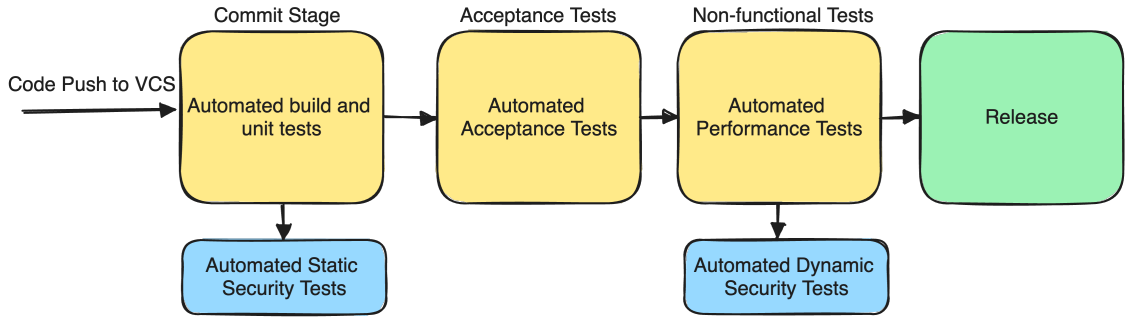

Figure 9. Enhanced Deployment Pipeline.

Importantly, the testing starts with static tests at the commit stage which shows the emphasis on shift-left security to provide fast feedback for the development team. Then the time-consuming dynamic testing is done later in the pipeline in a production-like environment, ideally the issues found in dynamic testing could be caught earlier with custom static security tests, the would be a practical example of shifting security left to improve time to feedback.

The static vs dynamic security tests can also be positioned into CI and CD categories as seen in Figure 10 below.

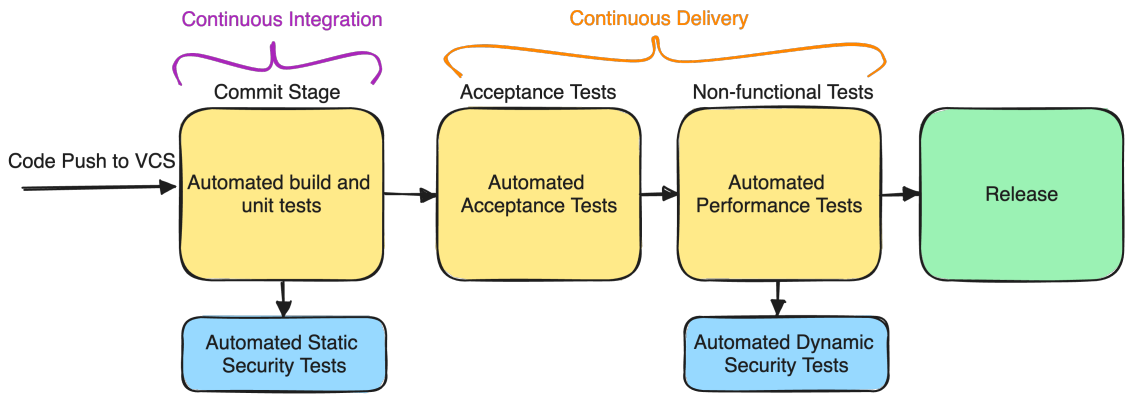

Figure 10. Continuous Integration and Continuous Delivery scoped in the Deployment Pipeline

In Figure 10, the CI and CD practices are roughly scoped above the practical steps in the deployment pipeline. The deployment pipeline should always be co-owned by the Development, Operations and Security teams in an organization practicing DevSecOps culture. Following the Test-Driven Security principle, a security test can be added by anyone working with the software being developed, and at any phase of the pipeline.

This allows people with different backgrounds and specializations to collaborate and share their knowledge to improve the overall quality and security posture of the software.

It's important to acknowledge that this way of integrating security testing into the software delivery and development process does require certain maturity from the organisation developing the software. Teams should be practicing the core practices of Continuous Integration and ideally Continuous Delivery to easily integrate the security checks and tests described in this section into their workflow. Next the element 2 of the initial proposal dives deeper into what kind of tests should be added and at what phase of the pipeline.

5.3.2 Element 2 of the Initial Proposal: Secure Software Development Lifecycle Phases

The second element of the framework aims to make it as easy as possible to identify when and where a tool or practice should be implemented and what outcomes should be expected. This second element acts as the core and the most content rich part of the framework implementation. The Secure Software Development Lifecycle Phases model is inspired by the lifecycle phases described in the CNCF Cloud Native Security Whitepaper (CNCF 2022). The model has been modified to make the model more aligned with continuous delivery concepts and thus also with the context of the case company. The modifications to the original model and the purpose of each phase are described next per lifecycle phase.

5.3.2.1 Develop and Integrate

In the original CNCF Cloud Native Security Whitepaper (CNCF 2022), the first lifecycle phase is called "Develop". In the original model the security checks are done with a pre-commit hook. However, pre-commit hooks are problematic because they cannot be enforced to be used by developers, since they are configured locally on developers' machines. Instead of pre-commit hooks, the framework recommends implementing these practices in the continuous integration stage in the pipeline. Because of this recommendation, the name of the lifecycle phase became "Develop and Integrate" to indicate the continuous integration phase of the deployment pipeline is in scope of this phase.

At this phase, the developer is working on code, but the scope is not just application code since with the platforms available today also many of the infrastructure and runtime requirements can be described as code or configuration. After the code is checked into the source code repository, it’s also possible to request a peer review on the changes. However not every change needs to be checked, or at least it should not be a blocking and gatekeeping activity that slows down the development flow.

Basically, at this phase for an automated security check the information available is text files such as source code or infrastructure as code configuration. These files can be analyzed by a static security analysis tool which can pick up obvious misconfigurations out of the box, and with custom tools or policies more advanced issues can be notified. Developing the more advanced checks could very well be done by a security expert on the subject matter.

The continuous integration stage sometimes called the “commit stage”, is recommended as the best way to implement these security checks so that they are truly done for every commit to the code base. It’s important that the person making the change to the code base should get fast feedback from the failing security check at the commit stage and then make a corrective action, even if that’s a rollback of the change, to fix the failing check. Ownership of this step can be shared, or it can be fully owned by the development team, the purpose is to make the person introducing the problem aware and responsible for the issue as soon as possible so that it does not come back as re-work later. There are many open-source tools available that can be used with default configuration at this phase and the ones recommended in the framework are all easy to use, helping to implement them. Instead of focusing on what tool is the best, it’s more important to just get the process in place.

5.3.2.2 Test and Distribute

In the original whitepaper CNCF Cloud Native Security Whitepaper (CNCF 2022), the second lifecycle phase is called “Distribute”. This name was found confusing and not descriptive of the full scope of the lifecycle phase. Thus, it was changed to “Test and Distribute” which better captures the fact that after the initial commit stage, there will be additional tests in a production like environment and long-running security tests can be introduced at this stage.

Although the software should already be sanity checked and built during the commit stage, the software is typically built again and packaged into a packaging format or an image format such as a container image. When this happens there are many things to consider from a security perspective, such as the base image used (in case of building a container image) and what known vulnerabilities might be present given all the dependencies of the software. It's also important to consider supply chain security best practices and produce metadata from the build, however those are discussed separately as part software supply chain security in the foundational security practices section since it's a large topic.

At this phase, the scanning for known vulnerabilities should be done since all dependencies should be present when a release candidate is built into an artifact. In addition to scanning for known vulnerabilities, also dynamic security scanners can be used to target the running software to check if the software can be compromised in some way. It's worthy to note that these tests are not sole responsibility of the developer team, especially at this stage all teams can collaborate to create a test suite that automatically determines whether the software (at this commit) can be released. Since these tests are executed in a production-like environment, it should be natural that the Operations and Security teams have collaborated to the setup of this environment and misconfigurations should be caught. There are many excellent open-source tools available for security scanning, and the framework recommends using tools that are easy to take into use to make the implementation effort in the beginning small and establish the foothold for this process in the deployment pipeline.

The actual act of releasing and deploying the software is left to the "Deploy" phase discussed next.

5.3.2.3 Deploy

The third lifecycle stage, "Deploy", is left named and scoped the same as in the original CNCF Cloud Native Security Whitepaper (CNCF 2022). This is a crucial step where the decision to deploy the software to production can be made manually or automatically (in case of continuous deployment). This is the final stage where so called "pre-flight" checks are run that should run quickly but ultimately enforce the policies in the production environment, blocking all insecure deployments.

In an ideal world, the pre-flight checks are mostly just criteria based on existing test results and verifications on the integrity of the software, introducing new gatekeeping checks at the deploy phase should be considered an anti-pattern since that makes the feedback loop slow and is against the shift-left principle.

The pre-flight check is in practice a double-check to verify that all configurations and artifacts are compliant with the policies set by Dev, Ops and Sec. Taking the DevSecOps driven deployment pipeline to its logical conclusion, this software release candidate is releasable if it passes the tests. No manual testing is needed afterwards even if the decision to release the software is manual, it should only be a push of a button.

There are many excellent open-source tools available to help implement the deploy phase. Again, the framework suggests tools that are easy to use and get into place, instead of tools with most features. There's also some contextual decision to be taken on technically how the pre-flight checks stage can be implemented given the technologies used by the customer, the framework tools depend on this context.

5.3.2.4 Monitor

In the original whitepaper CNCF Cloud Native Security Whitepaper (CNCF 2022), the fourth and final lifecycle phase is called "Runtime". In order to emphasize the main activity that the teams should be doing during runtime operations, it was decided to rename the lifecycle phase to "Monitor" in the framework.

During the monitor phase, the main source of information is different telemetry that is collected from the workloads, typical examples include metrics and logs. The most important practice is to create reports and alerts. Reports help to visualize the large amount of information and highlight possible security threats. Alerts that notify people should be used sparingly to avoid alert fatigue, but still they are important to setup to be notified about suspicious activities such as large amount of failed login attempts.

While this lifecycle phase is important for the security posture of any organization, it's not seen as the main area of focus for accelerating the security posture given the nature and scope of typical customer projects of the case company. The framework suggests

few tools in this space that are easy to use and get started with the process, but ultimately the content of the framework is quite light on this phase compared to earlier phases.

5.3.3 Element 3 of the Initial Proposal: Foundational Security Practices

The third element of the initial proposal of the framework includes three carefully selected foundational security practices: *Software Supply Chain Security*, *Secrets Management* and *Version Control System Security*. A lot goes into a holistic security strategy and many of the practices can be seen as foundational, but these three practices were identified as essential for the case company based on both the interest and relevancy given the typical customer assignment. Next the best practices and recommendations from the framework are discussed under each foundational security practice.

5.3.3.1 Software Supply Chain Security

As noted in the existing knowledge section of this thesis, software supply chain security is an important part of security posture for organizations. Frameworks such as SLSA and in-toto are still in their infancy and this field is evolving quickly, thus it's important for case company consultants to understand not just the current state, but also understand future roadmap of these frameworks and the larger ecosystem around them.

As discussed in the Test and Distribute lifecycle phase, the most fundamental part of software supply chain security happens there since the signing, SBOM generation and build provenance generation all happen at the build time when release candidate is packaged into an artifact. Also, importantly in the Deploy lifecycle phase there's a need to verify the integrity and source of artifacts before deploying them to a production environment.

Practically, there are three core practices that should be adopted by an organization that is producing software: (1) Signing, (2) SBOM, and (3) Build Provenance.

There are open-source tools and ecosystems around all these three concerns that are recommended in the framework for practical implementation. In addition, there's a deep dive on achieving the highest, level 3, SLSA framework v1.0 compliancy for an example artifact. The example also includes signing and SBOM generation as part of the

documented deep dive, serving as an example implementation for the consultants. Future roadmap for the supply chain security is looking good with vendors adding features to automatically generate SBOMs and packaging ecosystem tooling is starting to adopt the SLSA framework for producing and verifying provenance, along with signatures.

5.3.3.2 Secrets Management

Secrets management was found the most important foundational topic amongst the consultants in the current state analysis. Also, there are some clear best practices that were identified from the existing knowledge on secrets management.

One the most important practices is to adopt a dedicated secrets management system. This will help with practical issues in modern DevOps driven environments, such as secrets sprawl which can be mitigated by placing secrets into a central system.

Based on the suggestions during the proposal building there was additional fine tuning to the initial proposal. Decision was to focus on basic usage of a tool and drive the adoption with an approach that's easy to implement, instead of focusing on the advanced features of secrets management systems the aim is to simply have one in use. By utilizing a secrets management system or pushing the initiative that one should be in place, the consultants can set an example and help others adopt the tool by gaining practical experience on the available tools.

5.3.3.3 Version Control System Security

All of the case company's consultants work with version control systems daily, and it's also often highly coupled with the deployment pipelines, thus covering it in the security framework was seen essential.

OpenSSF released the SCM [Source Code Management] Security Best Practices guide during the writing process of this thesis, and it quickly became the reference existing knowledge used in the framework for what security best practices should be in-place. In addition the OpenSSF guide, the framework also recommends open-source projects for continuous monitoring and compliance auditing for the leading VCS platform GitHub. As

the recommended first then, when working in customer projects the consultants can make sure to configure their own repositories according to these best practices.

5.3.4 Element 4 of the Initial Proposal: Security Landscape

The initial proposal has a strong core that consists of the Secure Software Development Lifecycle phases, that is then supported by the overarching DevSecOps culture and guiding principles. The core sits on top of the three most important foundational security practices for the case company. During Data 1 and Data 2 discussions, it was essential to get input on how to scope the initial proposal. From the discussions regarding the scope, a diagram was created as an extra visual aid to map security practices with an example application stack.

As seen in Figure 11 below, the security landscape for a typical application stack is extensive:

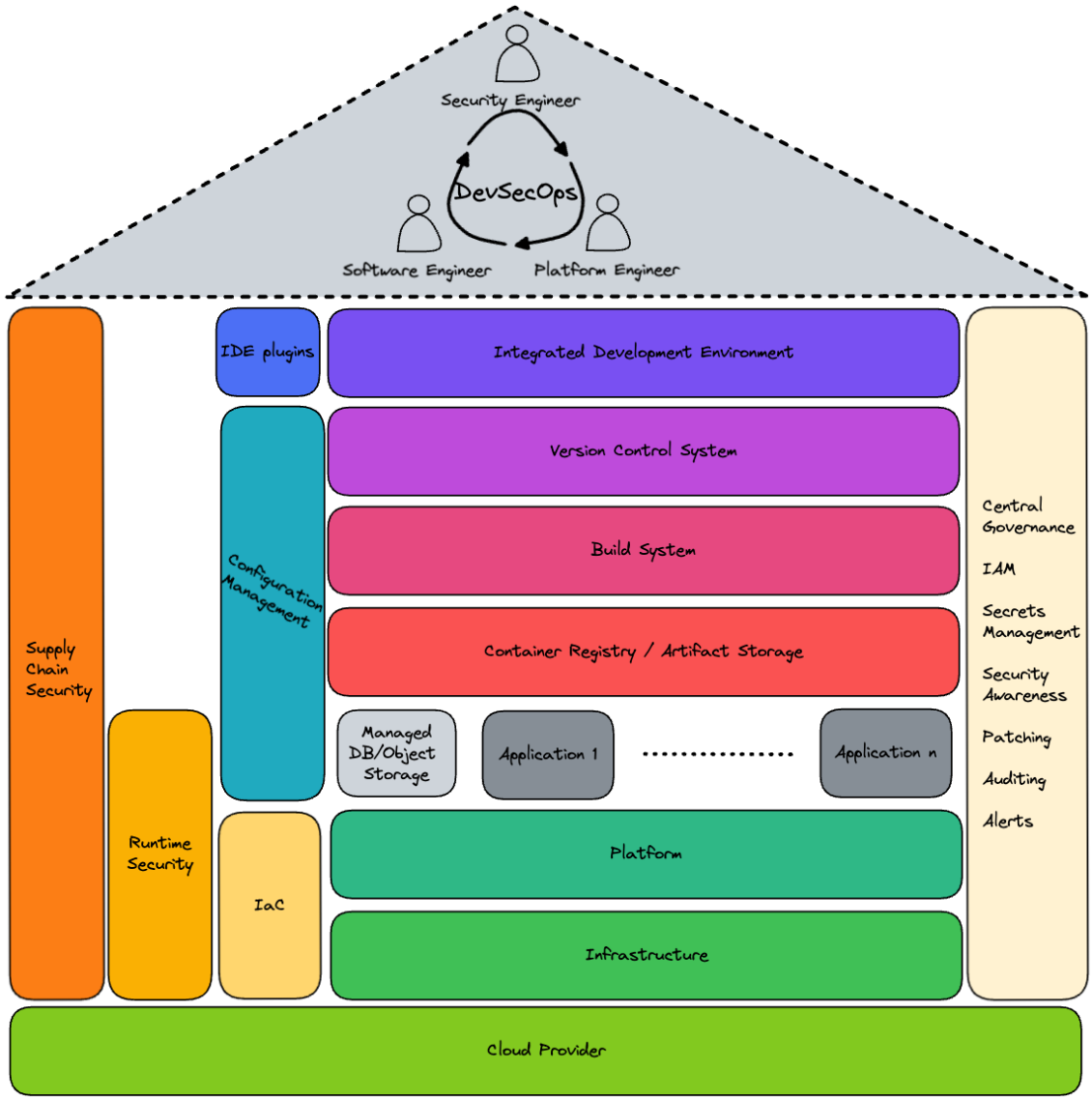

Figure 11. Security Landscape

Figure 11 shows the security landscape, with an example application stack illustrated in the middle, building on top of layers of infrastructure at the bottom and development tooling at the top of the applications themselves. This diagram was created to help map the framework elements to real world software development processes and a typical production environment, public or private cloud. Instead of only showing the framework as it's scoped, this diagram helps to also understand what's not covered in the framework as of now, as an example the diagram includes many practices in the right side (e.g. IAM, auditing, awareness) that are not explicitly covered in the framework to keep it focused on the case company context.

Summing up, the initial proposal leaves room for future improvement, but most importantly it has a core that is meant to get all the consultants onboard no matter what their current task is, there should be a relevant security practice they can take into use from the framework. Next, Section 6 focuses on validation of the initial proposal and then present the final version of the proposal.

6 Validation of the Proposal

This section discusses the validation stage and the third data collection round (Data 3) gathered from the validation comments and overall feedback from the stakeholders. At the end of this section, the final proposal is presented.

6.1 Overview of the Validation Stage

The initial proposal was a draft of the continuous security framework, which with elements focusing on addressing the weaknesses identified during the current state analysis. The framework draft was also informed by best practices from the conceptual framework that were fit for the context of the case company, and additionally, the suggestions gathered from the data collection (Data 2) done during initial proposal building.

The initial proposal included four elements: *DevSecOps Culture and Methods*, *Secure Software Development Lifecycle Phases* and *Foundational Security Practices*. Aim of the validation is to gather feedback on the initial proposal to develop a final proposal and then move on to complete the implementation of the first iteration of the framework. The validation was performed in a single session during an internal conference, with most of the case company experts present. This full validation process consisted of three steps:

First, a face-to-face presentation was given. During the presentation the internal website containing the draft of the initial proposal was shared and additional context was given verbally on each of the main elements and how to interpret the structure of the framework. Second, questions, comments and other feedback was gathered from free-form discussion in the room after the presentation. This feedback was recorded as field notes. Third, based on the feedback gathered there were developments to the initial proposal. After these developments were done the final proposal was created by updating the website. The next section discusses the comments and feedback gathered during the validation session and presents the developments that followed.

6.2 Developments to the Initial Proposal

This section discusses the comments and feedback received during the validation stage and points to further developments to the initial proposal. At the end of the section, the final proposal and implementation of the first iteration of the continuous security framework are presented. Table 4 below presents the comments and feedback gathered during the validation session, and the further developments to the initial proposal.

Table 4. Expert suggestions for the initial proposal.

	Element of the Initial Proposal	*Parts commented in Validation*	*Description of the comment/ feedback by experts*	*Development to the Initial Proposal*
1	Element 1: DevSecOps Culture and Methods	a) Methods; Test-Driven Security and Shift-Left Security	Test-Driven security is a helpful term with customer discussion on how we approach implementing security in practice.	Make sure the terminology is consistently used throughout the framework implementation, to make it part of the core 'language' when discussing security.
2	Element 2: Secure Software Development Lifecycle Phases	a) Develop and Integrate Phase	How exactly should the peer review be done, and when?	Peer review wasn't well defined, it was decided to drop it from the scope of the first iteration of the framework.
		b) Test and Distribute Phase	What about application security, could we scan the application in testing environment?	The question was about using dynamic application security testing tools, after a conversation it was decided these tools are not in scope due to required expertise. Instead a static analysis tool for application source code (gosec as an example) in the Develop and Integrate phase, since static analysis tools don't require as much expertise.
		c) Deploy Phase	The artifacts available between Deploy phase and the previous (Test and Distribute) phase are confusing.	Clarify the "What information and artifacts are available?" column by making it more high level and descriptive.
3	Element 3: Foundational Security Practices	a) Software Supply Chain Security	Can we take this into use for internal projects?	Align the technologies used in the software supply chain security examples with what is used for internal projects, to make it easy to adopt internally.

As seen in Table 4, there were several useful suggestions from the stakeholders during the validation session which led into further development of the initial proposal. In the next sub-sections the feedback on all four elements of the proposal will be discussed in detail.

6.2.1 Developments to Element 1: DevSecOps Culture and Methods

Element 1 of the initial proposal is less tangible compared to the technical practices described in the element 2 and element 3. As seen in table 4, the development to the initial proposal was to make sure the “Test-Driven Security” term is used consistently in the implementation of the framework, because stakeholders found that it’s descriptive of the core idea of how the framework aims to implement the security practices; using automated tests that are run continuously alongside development. There was no further feedback on the element 1. However, one of the comments gathered from the validation session might explain why:

> “A lot of time and effort has been put into the framework, now we just need to start using this.” (Consultant 1)

As the quote above mentions, the framework needs to be tested in practice and then the feedback from the consultants and customers can help to further validate the methods described in the element 1. Unfortunately, there was no time to conduct meaningful real-world testing of the security framework for this thesis that might have revealed further developments necessary. Positively, the framework is found useful based on the feedback and the consultants are eager to try out the various practices it describes, including the Test-Drive Security and Shift Left security methods described in the element 1 of the initial proposal.

6.2.2 Developments to Elements 2: Secure Software Development Lifecycle Phases

Element 2 of the initial proposal received the most scrutiny that led into further developments of the proposal out of the four elements.

First, there was a comment on how the peer review could be done during the Develop and Integrate lifecycle phase. During conversations one of the consultants expressed their view:

> “Security peer review process if definitely something we should establish, but I feel it might take some time to find a format that works for us, we will have to practice and iterate on this.” (Consultant 2)

The above quote summarises the conversation well. Overall there was a consensus that the practice is useful but it needs additional development and testing. The development that came out of this feedback was that the peer review was dropped from the scope of the first iteration of the framework, but there's still a placeholder for it in the website waiting for further development.

Second, there was a comment on what could be done with regards to application security during the Test and Distribute lifecycle phase. More specifically, if the framework describes some dynamic security scanner(s) that could be run against a running application. It was further discussed that such tools had been briefly tried during the initial proposal building when identifying tools to recommend in the initial framework. However, those tools were far and few between, and the from the ones tried the results were much harder to analyse without an application security expertise, thus such tools were not recommended in the initial proposal of the framework. Instead of using dynamic application security scanners, there was a development to the framework to make sure it includes at least one example of a static application security scanning tool which can be run already at earlier phase of the lifecycle and the results are easier to understand without application security expertise.

Third, the wording of the What information and artifacts are available? -column for the Deploy lifecycle phase was found confusing. One of the stakeholders gave the following comment:

> "Why are the provenance and SBOM artifacts in the Deploy phase, shouldn't those be in the previous stage?" (Consultant 3)

As the quote above mentions, the fact that SBOM and (SLSA) provenance are mentioned in both Test and Distribute and Deploy lifecycle phase is a source of confusion. This naturally led into a development of the wording used in the framework. The explicit mention of SBOM and a provenance artifact was replaced with a mention of "metadata", which is then further opened in the framework's practical content under this section.

6.2.3 Developments to Elements 3: Foundational Security Practices

The element 3 of the initial proposal was foundational security practices, it consists of three sub-elements: Software Supply Chain Security, Secrets Management and Version Control System Security.

Software Supply Chain Security received the most attention and feedback during the session. Many of the consultants though of concrete use-cases and benefits of generating SBOM, and there were in-depth discussion about what exactly is SLSA provenance and overall on the SLSA framework. One the stakeholders commented the following:

> “We should make our internal projects SLSA level 3 compliant!” (Consultant 4)

As seen from the quote above, there was lots of enthusiasm around the topic to put it into practice and learn more. This also lead into a further development of the initial proposal to make sure the content on software supply chain security align with the technologies used for the case company’s internal projects to make it easy to use the content as reference for implementation of the security practices for internal use.

The two other sub-elements of the foundational security practices: Secrets Management and Version Control System Security didn’t receive any feedback that would have lead into further developments.

6.2.4 Developments to Element 4: Security Landscape

The element 3 of the initial proposal was security landscape which is a diagram created to help map the overall elements of the security framework with an example application stack.

During the validation session there was no suggestions on how the security landscape could be improved or changes. The stakeholders found the element useful and supported the existing elements as it is in the initial proposal.

6.3 Final Proposal

This section presents the final proposal that was created after the validation stage suggestions were accounted for. The final proposal is also followed, in the next sub-section, by a brief discussion about the actual implementation of the first iteration of the continuous security framework website that was the main delivery for the case company.

Table 5. Final Proposal of the Continuous Security Framework

Culture and Methods	**Shift-left Security**			
	Test-Driven Security			
	DevSecOps culture			
Secure Software Development Lifecycle Phases	**Develop & Integrate**	**Test & Distribute**	**Deploy**	**Monitor**
What is the activity?	Writing code and configuration incl. deployment manifests and IaC.	Building, testing and distributing artifacts: container images, binaries, VM images, other packages.	Production deployment.	Monitoring production infrastructure and workloads.
What information and artifacts are available?	Static text files.	Software that builds and runs.	Software artifacts and their metadata, deployment configuration, test results.	Metrics, logs (app, network, OS), Cloud APIs and configuration.
What could we check for?	Known vulnerabilities in dependencies, insecure patterns in code/config.	Compliance, insecure runtime configuration, functionality, integrity.	Production readiness according to policies and compliance.	Suspicious activity, unstable software, over-privileged identities.
What is most important?	Fast feedback cycle, CI pipeline must complete in <5min. Catch known issues fast before developer context switches.	Software and configuration is tested and scanned. Provide timely feedback in <1h. Produce signed artifacts, SBOMs and provenance from build for distribution.	Verify and enforce integrity (signature, provenance), compliance (tags, labels etc.) and runtime configuration in pre-flight checks.	Alerts and reports.
How to check/achieve it? (tool/practice)	Static Analysis tools: trivy, tfsec, conftest, gosec etc. Custom policies for misconfigurations.	Automated tests. SLSA provenance, syft (SBOM), sigstore.	Admission controllers (Kyverno), conftest/OPA, custom checks.	Steampipe, Trivy Operator, Tetragon/Falco.
Foundational Security Practices	**Software Supply Chain Security**			
	Secrets Management			
	Version Control System Security			
Security Landscape				

As seen in Table 5 above, the final proposal is structurally unchanged from the initial proposal seen in Table 4. The validation stage provided good improvements and improved the clarity of the framework, while the overall shape and elements remained the same. In the next section the practical implementation of the framework is discussed.

6.4 Implementation

The objective of this thesis work was to create a practical continuous security framework which would provide consultants tools for effective customer communication and with effective security practices for implementation in projects. As an outcome on the thesis work, a website was created that goes into much more details on the elements and sub-elements of the framework than what is possible to describe in this thesis. Figure 12 below is a screenshot of one the pages of the website that was created.

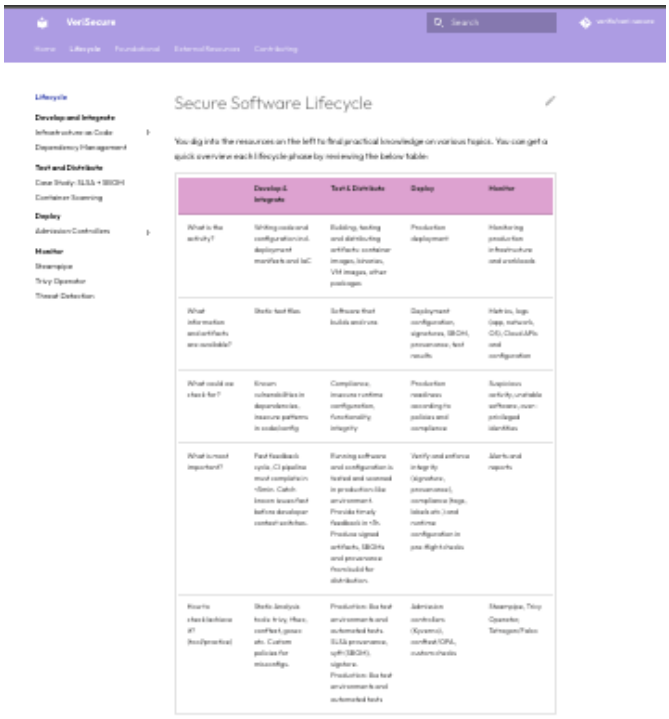

Figure 12. View of the Software Security Lifecycle Phases on the company´s website.

As seen in Figure 12 above, the Software Security Lifecycle Phases are included as a set of pages that go into lots of details on the different phases and the technical practices that are described only on high-level in this thesis. In addition to the lifecycle phases,

there are other tabs in the website that cover the other elements. Navigation bar of the website is shown in Figure 13 below.

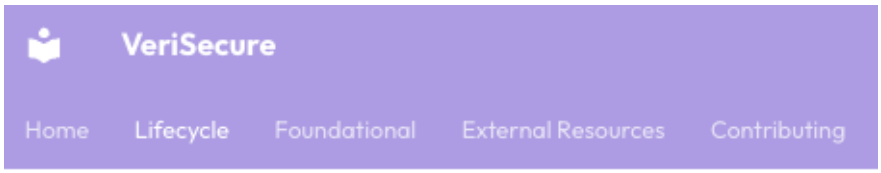

Figure 13. Navigation bar of the website.

As seen in Figure 13, the website clearly contains the elements 2 and 3 described in the final proposal, and less explicitly the element 1 is described in the home page since it's the main methodology and strategy supporting the technical implementation work.

Also as seen in Figure 13, the website was named as VeriSecure, following the internal naming conventions of the case company, for example the yearly internal conference is called VeriConf. There have been discussions on whether the name should be shortened to VeriSec, but no consensus has been reached at the time of writing this.

In addition to the website, also one blog post was released during the thesis writing process and likely in future there will be more blog posts about the technical topics inspired by the contents of the website. After more of the ideas on the website are put into practice, hopefully the website will keep growing and through future iterations of the framework maybe it will transform to either cover more ground or to focus on specific element(s) more and more. Best way to further iterate on the website is to look for opportunities in customer projects to apply a practice from the framework, and contribute back with feedback and practical knowledge gained from the implementation of the practice.

The following section will discuss the conclusion of the thesis. Section 7 below includes the executive summary, discussion of next steps for the framework, thesis evaluation and finally some closing words.

7 Conclusion

This section summarises the thesis results in an executive summary. In addition, the thesis is evaluated and concluded with some closing words.

7.1 Executive Summary

The objective of this thesis was to create a continuous security framework that would equip the case company consultants with tools for effective customer communication and, provide practical guidance on effective security practices that should be implemented in customer projects. Increased speed in software development and delivery process due to DevOps movement requires changes to the cybersecurity processes and practices in many organisations to make sure the increased speed of software delivery does not translate into decreased security posture. The case company focuses on DevOps and Cloud Architecture consulting services, and it's important that the consultants are well informed and up to date with suitable security best practices to help the specialised security experts and teams of customer organisations by being proactive and leading by example with regards to security practices in customer projects.

The research approach used in this thesis is applied action research and mainly qualitative research methods. This research approach was selected to arrive at clear results by implementing a practical change towards the business challenge without trying to generalise the results for different contexts. The qualitative methods were applied during three data collection rounds conducted during the thesis work, the data was collected from interviews, analysis of internal documents and systems, workshops and a survey.

The first data collection round (Data 1) was for the current state analysis which gathered information inside the case company on what and how security related tasks and practices are currently implemented in customer projects throughout the company. It was found that there is no structured process or documentation around security practices currently, so an analysis was needed on what is happening in projects to identify the current strengths and weaknesses, and areas for improvement. After analysing the current state, the following key focus areas were selected for a literature review to identify suitable best practices that could be applied in the context of the case company: (1)

DevSecOps, (2) Software Security Lifecycle Phases, (3) Software Supply Chain Security, (4) Secrets Management and finally (5) Securing Source Code Management. The outcome of the literature review was a conceptual framework that touches on best practices for people and processes as well as for implementation.

An initial proposal was created based on the focus areas identified through the current state analysis and the best practices found from literature that were distilled into the conceptual framework. This initial proposal is the first draft of the elements of the continuous security framework. The first draft was then presented to the case company experts for feedback and the initial proposal was further developed into a final proposal based on this feedback. During the proposal building the scope and shape of the continuous security framework was adjusted and made sure that the framework is fit for the case company context, instead of being generalised. The proposed continuous security framework consists of four elements: (1) an overarching DevSecOps strategy and methods to guide the implementation and decision making, (2) secure software lifecycle phases as a core practical guidance for steps to implement in each phase, (3) foundational security practices to help secure foundation on which the lifecycle phases execute, and (4) security landscape diagram that helps to map the security practices with an example application development and delivery stack. Following the final proposal the first iteration of the framework was implemented as an internal website that is a collection of practical guides and examples on the elements described in this thesis.

The proposal was validated in a face-to-face session with all the key stakeholders present. The website was shown to stakeholders and then most points of interest in the content and the future were discussed. Overall, the feedback was that there's lots of effort put into the framework, but it needs to be put into implementation to figure out the next steps and further iterate.

The continuous security framework built was well received and the data collection process also helped to raise awareness and solidify the need for explicit focus on security in customer work. Some of the practices from the framework have already found their way into internal projects and hopefully contributions from various case company consultants will help to shape the future iterations of the framework to something even more fitting for the case company needs.

7.2 Thesis Evaluation

The need for the thesis was to create a security framework that integrates well with the specialisation area (DevOps/Cloud) that the case company provides expert services in. The original spark for the idea came from a small amount of experience using open-source security scanning tools in recent projects, they were easy to use, and the feedback was positive and useful. However, running a few tools in a CI pipeline versus designing a full framework for “all projects” of the case company was a bigger challenge. A lot of the existing knowledge studied was written by full-time security professionals, when analysing the best practices from those sources it was difficult to judge if it fits in the context of the case company or if it’s too grandiose and specialised.

When moving into the practical work on the proposal, there was a constant problem with trying to scope the framework into concrete and practical elements. Recommended tools must be easy to use and quick to learn, and the way to integrate them into the existing workflows must be seamless. For example, these are some of the questions that kept coming up: Can we make sense of the output from the tool, or does it require expertise? What is applicable in the typical role the consultants are in customer assignments? How would the customer react to proposing implementing a tool X for security enhancements? These were hard questions because the company does not focus explicitly on security, the scope of the recommended security practices needs to be carefully balanced with the expectation of customers on time spend on security when it’s not the main value of the delivery. Additionally, there was a risk when starting the work that the consultants don’t understand or see the value in the practices that the framework would promote. Fortunately, in the current state analysis it was clear that the consultants do care about security topics, a lot, and the initiative was well received although scope kept changing in the initial stages.

Ultimately, the scope of the framework in the end is still arguably too wide, some of the elements did not receive much scrutiny during the validation and it could have been great to reduce the scope more aggressively before implementation. The framework could have been just a small core of the very essential practices and guidance. This small core would have been much easier to grasp for the consultants and easier to put in practice to gather feedback, and then iterate in small steps to increase the scope of change the elements based on reflections on practical work. Some elements that received most attention, notably software supply chain security, could have worked as the main topic

of the thesis alone, but that space is moving fast and putting in lots of effort now might not be a great investment in the longer run as the ecosystems and tools are still quickly evolving.

7.3 Closing Words

During the thesis work, there were a lot of good conversations with colleagues, it was surprising how much people really care about security. Idea for the framework was to find ways to make these consultants effective and efficient when looking to apply security in their existing or future projects, or in other words, make it as easy as possible for the peer consultants to add security into the mix, without adding a lot to their cognitive load. The author learned a lot from the thesis and will be there for the peer consultants to help with security advice and coaching.

During the research, development and practical implementation work done for the thesis, the author also became more and more conscious of the amount of complexity in today's cloud native application environments. Securing all the layers of infrastructure is hard, on the other hand, failing to do so in one place can lead into an easy opening for a cybercriminal. Perhaps as an industry we would do everyone involved a favour if we would stop adding complexity and just worked for a while on what we already have, making that easy to use, stable, and secure by default before jumping into the next new hype train.

References

Adams, John & Khan, Hafiz T. A. & Raeside, Robert 2013. Research Methods for Business and Social Science Students. 2nd edition. New Delhi: SAGE Publications

Agilemanifesto.org 2001. Agile Manifesto. Retrieved 17/11/2023. https://agilemanifesto.org/

Anchore 2023. Syft, CLI tool and library for generating SBOM. Retrieved 2/11/2023. https://github.com/anchore/syft

Bird, Jim 2016. DevSecOps. Chapter 3. Keys to Injecting Security into DevOps. (e-book). Sebastopol: O'Reilly Media Inc. Retrieved 2/11/2023. https://learning.oreilly.com/library/view/devopssec/9781491971413/

CNCF 2022. Cloud Native Security Whitepaper, version 2. Retrieved 2/11/2023. https://github.com/cncf/tag-security/blob/main/security-whitepaper/v2/CNCF_cloud-native-security-whitepaper-May2022-v2.pdf

CNCF 2018. CNCF Cloud Native Definition v1.0. Retrieved 2/11/2023. https://github.com/cncf/toc/blob/main/DEFINITION.md

CNCF 2021. CNCF Software Supply Chain Best Practices. Retrieved 2/11/2023. https://github.com/cncf/tag-security/blob/main/supply-chain-security/supply-chain-security-paper/CNCF_SSCP_v1.pdf

Continuousdelivery.com 2010. Deployment Pipeline Anti-Patterns. Blog post 09/2010. Retrieved 1/11/2023. https://continuousdelivery.com/2010/09/deployment-pipeline-anti-patterns/

Cyberark 2023. Secrets Management. Retrieved 17/11/2023. https://www.cyberark.com/what-is/secrets-management/

Dadgar, Armon 2018. What is "secret sprawl" and why is it harmful? Retrieved 2/11/2023. https://www.hashicorp.com/resources/what-is-secret-sprawl-why-is-it-harmful

Davis, Jennifer & Daniels, Katherine 2016. Effective DevOps. Building a Culture of Collaboration, Affinity, and Tooling at Scale. Sebastopol: O'Reilly Media Inc

DeHamer, Brian & Harrison Philip 2023. Introducing npm package provenance. Blog post 19/04/2023. Retrieved 2/11/2023. https://github.blog/2023-04-19-introducing-npm-package-provenance/

Dotson, Chris 2019. Practical Cloud Security: A Guide for Secure Design and Deployment. Sebastopol: O'Reilly Media, Inc

European Commission 2022. Cyber Resilience Act. Retrieved 2/11/2023. https://digital-strategy.ec.europa.eu/en/library/cyber-resilience-act

Farley, David & Humble, Jez 2010. Continuous Delivery. Reliable Software Releases Through Build, Test and Deployment Automation. Boston: Pearson Education, Inc

Forsgren, Nicole & Humble, Jez & Kim, Gene 2018. Accelerate. Building and Scaling High Performing Technology Organizations. Portland: IT Revolution Press LLC

GitGuardian 2023. GitGuardian. Product website. Retrieved 17/11/2023. https://www.gitguardian.com/

GitGuardian 2023a. Secrets Detection. Retrieved 17/11/2023. https://www.gitguardian.com/monitor-internal-repositories-for-secrets

GitHub 2023. GitHub security features. Product documentation. Retrieved 2/11/2023. https://docs.github.com/en/code-security/getting-started/github-security-features

GitLab 2023. The Ultimate Guide to Securing your Code on GitLab.com. Retrieved 17/11/2023. https://about.gitlab.com/blog/2023/05/31/securing-your-code-on-gitlab/

Gray, Jim & Vogels, Werner 2006. A Conversation with Werner Vogels. Interview 30/06/2006. Retrieved 2/11/2023 https://queue.acm.org/detail.cfm?id=1142065

Guac.sh 2023. GUAC, Know your software supply chain. Retrieved 2/11/2023. https://guac.sh/

HashiCorp 2023. Vault, manage secrets and protect sensitive data with Vault. Retrieved 2/11/2023. https://www.vaultproject.io/

Kananen. 2013. Design Research (Applied Action Research) as Thesis Research. A Practical Guide for Thesis Research. Jyväskylä: Publication of JAMK University of Applied Sciences

Kim, Gene & Humble, Jez & Debois, Patrick & Willis, John & Forsgren, Nicole 2021. The DevOps Handbook. How to Create World-Class Agility, Reliability, & Security in Technology Organizations. 2nd edition. Portland: It Revolution Press LLC

Linux Foundation 2023. Sigstore. Mak Retrieved 2/11/2023. https://www.sigstore.dev/

Linux Foundation 2023a. in-toto Framework. Retrieved 2/11/2023. https://in-toto.io/

Linux Foundation 2023b. Tekton Chains. Supply Chain Security in Tekton Pipelines. Retrieved 2/11/2023. https://github.com/tektoncd/chains

NTIA 2020. Software Bill of Materials (SBOM). Retrieved 2/11/2023. https://www.ntia.gov/files/ntia/publications/sbom_overview_20200818.pdf

OpenSSF 2023. OpenSSF Releases Source Code Management Best Practices Guide. Blog post 14/09/2023. Retrieved 2/11/2023. https://openssf.org/blog/2023/09/14/openssf-releases-source-code-management-best-practices-guide/

Podjarny, Guy 2021. Cloud Native Application Security. Chapter 1: Accelerating Technology Delivery. (e-book). Sebastopol: O'Reilly Media Inc. Retrieved 23/10/2023. https://learning.oreilly.com/library/view/cloud-native-application/9781098105631/

Sheridan, Kelly 2020. Twilio Security Incident Shows Danger of Misconfigured S3 Buckets. News article 23/07/2020. Retrieved 20/11/2022. https://www.darkreading.com/cloud/twilio-security-incident-shows-danger-of-misconfigured-s3-buckets

SLSA.dev 2023. Supply chain threats. Retrieved 31/10/2023. https://slsa.dev/spec/v1.0/threats-overview

SLSA.dev 2023a. About SLSA. Retrieved 31/10/2023. https://slsa.dev/spec/v1.0/about

SLSA.dev 2023b. Software attestations. Retrieved 31/10/2023. https://slsa.dev/attestation-model

Sonatype 2023. State of the Sotware Supply Chain report 2023. Retrieved 2/11/2023. https://www.sonatype.com/state-of-the-software-supply-chain/open-source-supply-and-demand

Synopsys 2023. Open Source Security and Risk Analysis report. Retrieved 31/10/2023. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html

Truffle Security 2022. An API Worm In The Making: Thousands Of Secrets Found In Open S3 Buckets. News article 03/08/2022. Retrieved 20/11/2022. https://trufflesecurity.com/blog/an-api-worm-in-the-making-thousands-of-secrets-found-in-open-s3-buckets/

Vehen, Julien 2018. Securing DevOps. Security in the Cloud. Shelter Island: Manning Publications Co

White House 2021. Executive Order on America's Supply Chains. Retrieved 2/11/2023. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/02/24/executive-order-on-americas-supply-chains/

White House 2021. Executive Order on Improving the Nation's Cybersecurity. Retrieved 2/11/2023. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

Appendix 1: Current State Analysis Survey

1. Which of these security controls do you think are most important? (top 3-5)

2. Out of the ones you chose, would you like to prioritise them?

3. Out of the ones you chose or any others, which would be the easiest one to implement?

4. What's stopping you from doing these things as part of every day project work?

5. What tools would you use to implement controls such as static code scans and IaC policies?

6. Should we explicitly include security into the design and implementation of customer work?

7. Do you think it's possible to create a structured approach that can be useful in any project?

8. Anything else you'd like to mention around security?