



École Polytechnique Fédérale de Lausanne

Designing and Integrating Secure Software Development Practices  
in Multinational Insurance Companies

by Yanis De Busschere

Master Thesis

Approved by the Examining Committee:

Prof. Dr. sc. ETH Mathias Payer  
Thesis Advisor

Dr. Giulia Traverso  
External Expert

Dr. Giulia Traverso  
Thesis Supervisor

EPFL IC IINFCOM HEXHIVE  
BC 160 (Bâtiment BC)  
Station 14  
CH-1015 Lausanne

August 30, 2024

# Abstract

This project focuses on designing and integrating secure software development practices within a multinational organization, addressing technical problems and non-technical factors—such as communication, ownership, and aligning security with business objectives—where they intersect with technical efforts. Conducted during a six-month internship at Ernst & Young AG (EY), the project involved designing, integrating, implementing, documenting and on-boarding security practices for a multinational insurance company. The project aimed to ensure compliance with numerous security requirements across the software development lifecycle and was divided into three subprojects. The first focused on the management of third-party libraries, aiming to secure third-party components used in development. The second addressed transitioning a secret management solution from Proof Of Concept to full operational deployment, facing challenges due to lack of communication and clear ownership. The third subproject involved developing a basic framework for threat modeling and establishing reference security architectures, balancing high implementation costs with compliance needs. Key results demonstrated that while technical solutions are essential, addressing non-technical issues like communication, ownership, and balancing business needs is also important to overcome the complexities of implementing secure software development practices in multinational organizations.

# Contents

<b>Abstract (English/Français)</b>	<b>2</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Motivation & Opening Remarks . . . . .	5
1.2 Context: Transformation Program of a Multinational Organization . . . . .	6
1.2.1 Overview of Insurance & CO . . . . .	7
1.2.2 The Transformation at Insurance CH . . . . .	7
1.2.3 Structure of Insurance CH . . . . .	8
1.3 Overview & Objectives of the project . . . . .	8
1.3.1 Subproject 1: Third-Party Libraries Management . . . . .	9
1.3.2 Subproject 2: Secret Management . . . . .	10
1.3.3 Subproject 3: Threat Modeling and Reference Security Architectures . . . . .	11
<b>2 Third-Party Library Management</b>	<b>14</b>
2.1 Requirements . . . . .	14
2.2 Current State & Tools . . . . .	15
2.3 Initial Solutions . . . . .	16
2.3.1 Implementation . . . . .	17
2.3.2 Onboarding and Maintenance Processes . . . . .	19
2.4 Final Solutions . . . . .	19
2.4.1 Implementation . . . . .	20
2.4.2 Process: Onboarding New Third-Party Dependencies . . . . .	22
2.4.3 Process: Continuous Maintenance . . . . .	23
2.4.4 Process: Annual Maintenance . . . . .	23
2.4.5 Drawbacks and Benefits . . . . .	24
<b>3 Secret Management</b>	<b>25</b>
3.1 Requirements . . . . .	26
3.2 Solution . . . . .	26
3.2.1 Overview . . . . .	26
3.2.2 Seperation in Namespaces . . . . .	27
3.2.3 Authentication for Users . . . . .	28

3.2.4	Authentication for Applications . . . . .	30
3.2.5	Adding Secrets . . . . .	30
<b>4</b>	<b>Threat Modeling and Reference Security Architectures</b>	<b>33</b>
4.1	Threat Modeling . . . . .	33
4.1.1	Theoretical Background . . . . .	33
4.1.2	Requirements . . . . .	34
4.1.3	Solution . . . . .	34
4.2	Reference Security Architectures . . . . .	59
4.2.1	Theoretical Background . . . . .	59
4.2.2	Requirements . . . . .	59
4.2.3	Solution . . . . .	60
<b>5</b>	<b>Conclusion</b>	<b>66</b>
5.1	Summary of Results . . . . .	66
5.2	Key Insights Across the Project . . . . .	67
5.3	Next Steps . . . . .	68
	<b>Bibliography</b>	<b>69</b>

# Chapter 1

## Introduction

### 1.1 Motivation & Opening Remarks

In 2023, the global cost of cybercrime reached a staggering \$8.15 trillion [1]. To put this into perspective, the same year, the global GDP was approximately \$105 trillion [2], meaning that the cost of cybercrime was nearly 7.8% of the global GDP. Remarkably, it means that if we considered cybercrime losses as a national GDP, this hypothetical economy would rank as the world's third-largest [2]. Often underestimated, this figure surpasses other significant global losses that are typically assumed to be among the highest. For instance, the global economic losses from natural disasters in 2023 amounted to \$380 billion [3], a figure that pales in comparison to the financial damage caused by cybercrime.

However, these financial repercussions are just the tip of the iceberg. The reputational damage to businesses and the long-term viability of affected organizations are equally concerning. According to the S-RM 2022 report, over 30% of respondents indicated that their business had suffered reputational damage as a result of a cyber incident [4], while a quarter reported losing the business altogether.

Despite the staggering financial and reputational cost of cybercrime, many organizations remain alarmingly immature in their cybersecurity practices. A 2021 survey from McKinsey revealed that 70% of companies did not have an established operating model and organization to professionalize the cybersecurity function. Nearly none of them had advanced, risk-based approach where they manage and measure security and privacy controls in an enterprise-risk framework, set risk-appetite thresholds, and include all stakeholders in the cybersecurity operating mode [5].

Several challenges contribute to this immaturity. Insufficient budget and resources are the primary obstacles, with 31% of business leaders citing budget constraints as a significant barrier to improving their cybersecurity posture [6]. Secondly, the shortage of skilled cybersecurity profession-

als is a persistent challenge, with a global shortfall of 4 million cybersecurity professionals reported in 2023 [7]. Finally, the increasing complexity of IT environments, with the proliferation of cloud services and mobile devices, adds another layer of difficulty [8].

One of the most critical consequences of these challenges is the proliferation of vulnerabilities at the software level, which have become a primary target for cybercriminals. According to the Verizon 2024 Data Breach Investigations Report, 14% of all breaches involved the exploitation of vulnerabilities as an initial access step. Moreover, 15% of breaches involved a third party or supplier, such as software supply chains, hosting partner infrastructures, or data custodians [9].

Not only are these attacks more and more prevalent, they also lead to some of the most significant and damaging breaches in recent history. The SolarWinds attack, for example, saw cybercriminals infiltrate a widely-used IT management software, leading to the compromise of numerous government and private sector networks. This breach alone is estimated to have impacted 18,000 organizations, including multiple U.S. government agencies [10]. Similarly, the Log4j vulnerability, one of the most severe in recent memory, was reported to be present in over 93% of enterprise cloud environments, exposing countless systems to potential exploitation [11].

Given the increasing threats posed by software vulnerabilities, organizations must prioritize the integration of secure software development practices into their cybersecurity strategies. Implementing these practices requires the adoption of new security solutions, which in turn presents a variety of challenges. These include ensuring compatibility with existing systems, managing the complexity of diverse software environments, and maintaining system performance while enhancing security.

This thesis will focus on the design and implementation of these security solutions, highlighting the technical challenges. While the emphasis is on overcoming technical obstacles in the design, such as the integration and security enhancement, the thesis will also address non-technical factors—such as communication, ownership, and aligning security with business objectives—where they intersect with technical efforts.

## **1.2 Context: Transformation Program of a Multinational Organization**

The work of this master's project was conducted during a six-month internship at Ernst & Young AG (EY) as part of a cyber transformation program for the Swiss regional branch of a multinational insurance company. This transformation program, which has been ongoing since February 2023 and is projected to continue until March 2026, focuses on enhancing the secure software development processes within the Swiss regional branch. For confidentiality reason, we will refer to the global organization as *Insurance & Co* and regional branch as *Insurance CH*.

### **1.2.1 Overview of Insurance & CO**

Insurance & Co is a prominent player in the insurance industry, providing a wide range of insurance products and services across various markets. Given the nature of its business, the company handles vast amounts of sensitive personal and financial information, making robust information security practices crucial. In response to the evolving cybersecurity landscape, Insurance & Co has developed an Information Security Policy specifically addressing Secure Software Development.

The Information Security Policy on Secure Software Development, instituted by Insurance & Co's global information security division, outlines comprehensive requirements that regional branches must comply with. The policy is structured into several categories, each detailing specific aspects of secure software development. For instance, it includes requirements for container security, secure coding practices, regular security assessments, and application operation.

All regional branches, including Insurance CH, must comply with this information security policy to ensure that they adopt and maintain mature and secure software development practices.

### **1.2.2 The Transformation at Insurance CH**

The overarching goal of this transformation program is to transform Insurance CH's software development processes to align with the internal security policy mandated by Insurance & Co.

However, to achieve compliance with this policy, Insurance CH requires an extensive cyber transformation. This transformation entails a overhaul of the entire existing software development lifecycle.

The transformation at Insurance CH is inherently structured into a series of distinct subprojects, each designed to tackle specific category of requirements from the Information Security Policy. This segmented approach allows for a focused and systematic transformation of Insurance CH's software development practices, ensuring that all aspects of the policy are comprehensively addressed. There are three different types of issue that these subprojects face :

- **No Existing Solutions:** In categories where no previous secure software development practices were in place, we introduce new processes and tools to establish a foundation for security. This involves adopting industry-standard practices and integrating them into the development lifecycle.
- **Incomplete Solutions:** For categories where existing practices were identified but found to be incomplete, we enhance and expand these solutions to meet the requirements of the ISP. In this thesis, this included moving from the proof-of-concept stage to finalization and widespread adoption. .

- **Undocumented Practices:** In cases where practices were being followed but were not formally documented or auditable, we focused on creating detailed documentation and ensuring that these practices were standardized and consistently applied across all projects. This included formalizing procedures and establishing clear guidelines for all development activities.

By addressing these three type of issues for each category, the transformation program aims to significantly elevate the security and maturity of software development practices at Insurance CH, ensuring alignment with Insurance & CO's information security standards.

### 1.2.3 Structure of Insurance CH

Insurance CH is structured into several specialized teams, each playing a critical role in driving the transformation forward:

- **Development Teams:** These teams are responsible for the implementation and enhancement of business applications. They play a crucial role in integrating secure coding practices and adopting the new security tools and processes introduced by the transformation program.
- **DevOps Team:** This teams focus on automating the software development lifecycle to increase efficiency and reliability. As part of the transformation program, these teams are tasked with integrating security measures into the continuous integration and continuous deployment (CI/CD) pipelines, ensuring that security is embedded in every stage of software life cycle.
- **Enterprise Architecture Team:** This team oversees the alignment of business applications with the organization's business needs. They ensure that the software development processes and outcomes align with the strategic goals of the company. The Enterprise Architecture team works closely with both the development and DevOps teams.

## 1.3 Overview & Objectives of the project

The work of this master's project focused on three critical subprojects within the broader cyber transformation program at Insurance CH. Each subproject aimed to address specific category of the Information Security Policy and elevate the secure software development practices within the organization. This section gives an overview of each subproject.



### 1.3.1 Subproject 1: Third-Party Libraries Management

This project focused on establishing a secure and compliant framework for managing third-party libraries within Insurance & CO's software development processes. The initiative was driven by the need to address significant security vulnerabilities identified during a gap analysis, which revealed that developers were freely downloading and using libraries and plugins without any formal approval or oversight. This situation not only exposed the organization to potential security risks[9], such as the introduction of vulnerabilities and legal issues related to licensing, but also highlighted a critical compliance gap with the organization's Information Security Policy.

Despite having the appropriate tools and infrastructure, such as Jenkins [12] for building and deploying and Artifactory [13] as a central repository manager for artifacts, there was a lack of control. Artifactory was configured to mirror most major public repositories, including Maven and NPM [14], which each contains millions of unchecked packages. Consequently, even though the Jenkins Nodes [15] were restricted from directly accessing the internet, developers still had access to unverified packages from these repositories, posing a risk of integrating vulnerable components into the development lifecycle. Additionally, there were no processes in place for adding and reviewing dependencies.

To address these vulnerabilities, a solution was developed that allowed the Artifactory server to maintain its connection to public repositories while introducing enhanced security measures through JFrog Xray [16] and well-defined processes that can be audited.

The solution was comprised of the following components:

- **Centralized Repository Management:** The solution retained the use of JFrog Artifactory as the central repository for all third-party dependencies, and kept access to public repositories. This kept the operational and initial implementation burdens low.
- **Security and Compliance via JFrog Xray:** Xray was integrated and configured to automatically scan all dependencies, both direct and transitive, within Artifactory. It ensured that only those libraries without critical or high vulnerabilities could be used, aligning with the organization's compliance requirements.
- **Onboarding Process:** The solution introduced an onboarding process for new third-party components. The process ensures that developers go through and document an onboarding checklist. If the library passes the onboarding checklists and gets accepted by Xray, it is approved for use. Otherwise, a risk acceptance must be made.
- **Maintenance Processes:** The project also defined two maintenance processes.
  - **Continuous Maintenance:** JFrog Xray is configured to monitor and scan all third-party libraries for new vulnerabilities in real time. When a new vulnerability is detected in any

dependency, the relevant development teams are immediately notified. It is then the responsibility of these teams to assess and address the vulnerability promptly, whether by updating the affected component, applying a workaround, or escalating the issue for further review and risk acceptance.

- **Annual Maintenance:** An annual clean-up and risk assessment process is conducted to ensure long-term security and compliance. This includes removing deprecated or unused components, reassessing the risks associated with current libraries, and exporting and documenting a Software Bill Of Materials (SBOM) for each application.
- **Initial Risk-Acceptance:** Given the extensive number of third-party dependencies already in use at the start of the project, an initial risk acceptance process was necessary.

In conclusion, this project addressed the critical vulnerabilities identified in the gap analysis by establishing a secure and scalable framework for managing third-party libraries. By balancing the need for security with operational efficiency, the organization was able to enhance its software development processes, reduce the risk of introducing vulnerabilities, and ensure alignment with its Information Security Policy.

### 1.3.2 Subproject 2: Secret Management

This subproject focused on establishing a compliant and standardized approach to managing secrets, such as passwords, API keys, and encryption keys, within Insurance CH's applications.

The DevOps Team at Insurance CH had previously purchased a license for HashiCorp Vault [17] and conducted some initial tests. However, due to a lack of assigned responsibility, the solution remained incomplete and unused, leaving development teams unaware of its existence. As a result, secrets were often hardcoded directly into applications, which not only created significant security risks[18] but also left a compliance gap in how sensitive information was managed across the organization.

This subproject took charge of completing the solution and defining the operating model for HashiCorp Vault to ensure it met compliance requirements. The final implementation involved several key components:

- **Namespace Structure:** Vault was configured with a namespace structure that provided logical separation of environments and development teams. Each team or application domain was allocated distinct namespaces for engineering, non-production, and production environments, following a standardized naming convention. This approach allowed teams to manage their secrets independently while ensuring that environments were isolated and protected according to their maturity level.

- **Role-Based Access Control (RBAC):** Integration with Azure Active Directory [19] was established to manage access rights using dedicated security groups. Each namespace had specific Azure AD groups controlling read, write, and administrative access, ensuring that only authorized users could access or modify secrets.
- **Authentication and Access Management:** Developers accessed HashiCorp Vault through a secure web interface using OIDC authentication [20], linked to Azure AD.
- **Secrets Management Processes:** Detailed processes were defined for adding, updating, and deleting secrets within the Vault. These processes ensured that secrets were managed in a controlled and compliant manner, reducing the risk of misconfiguration. The use of the Key-Value (KV) secrets engine, particularly in its versioning mode (KV2), enabled secure and traceable management of secrets, allowing for different versions to be maintained and audited.
- **Integration with Development Pipelines:** The solution was integrated with Kubernetes [21] clusters via the Vault Container Storage Interface (CSI) driver [22], enabling containerized applications to securely access secrets. Automated Jenkins pipelines were set up to manage the registration of clusters and creation of Secret Provider Classes, streamlining the process for developers and ensuring that secrets could be accessed securely during application runtime.

By finalizing and deploying this comprehensive solution, the project ensured that Insurance CH met all necessary compliance standards for secret management and provided a scalable and maintainable framework.

### 1.3.3 Subproject 3: Threat Modeling and Reference Security Architectures

The project's primary objective was to develop and implement solutions that would satisfy two closely related sets of requirements from the Information Security Policy: Threat Modeling and Reference Security Architectures. Despite their distinct requirements, these categories were treated together due to their shared challenges and overlapping nature. Both aimed to improve the security posture of applications and protect sensitive data by identifying, assessing, and mitigating potential threats early in the development process and both used a mix of architecture diagrams and data flow diagrams.

#### Threat Modeling

Threat modeling can be time-consuming and challenging, especially for those new to the practice [23]. Indeed, threat modeling requires a thorough understanding of potential threats, system architecture, and security principles. To address these challenges, the project aimed to provide a suite of architecture building blocks, with their associated threats and security measures, that developers can use to quickly model their applications and identify relevant threats. By offering

these predefined building blocks, the project lowers the entry barrier for developers to engage in threat modeling.

These architecture building blocks focus on common application scenarios, such as Authentication, Internal Data Store and Secret Management. They all include a technical diagrams, examples of threats using STRIDE [24], and a mapping to the security measures. Each building block was designed to provide a comprehensive view of an application's architecture from the angle of a specific feature, highlighting critical assets, potential threat actors, attack paths, and countermeasures.

To provide more value, another key point was the integration of threat modeling architectures with the existing risk assessment practices. Indeed, all applications were already required to go through a yearly Overarching Risk Assessment. The Overarching Risk Assessment evaluates the inherent risk associated with an application to propose relevant mandatory security measures. However, the link between the inherent risk and the security measures was never made clear. The architecture building blocks were designed to align and provide a clear link between the results of the risk assessment and the mandatory security measures that were already required. By serving as a bridge between the inherent risks identified in the risk assessment and the mandatory security measures, the methodology offered developers a clear understanding of potential threats to their applications and the corresponding security measures required to mitigate these threats.

Finally, to facilitate practical application, the project outlined clear steps for developers to follow. These steps included selecting and tailoring the appropriate building blocks to their specific applications, identifying relevant threats, and implementing the corresponding security measures. The methodology emphasized the importance of understanding and mitigating threats early in the development process to prevent security issues from becoming costly vulnerabilities later on.

By focusing on the development of building blocks, the project provided a rapid path to compliance while laying the groundwork for future, more detailed threat modeling.

### **Reference Security Architectures**

In conjunction with threat modeling, the development of Reference Security Architectures was aimed at providing a set of reference application architectures that developers and architects can utilize at the beginning of new projects or when implementing new features. These templates were intended to not only simplify the development process but also facilitate security considerations during the design phase.

Similarly to the part on Threat Modeling, this part's primary goal was not to create a comprehensive library of reference architectures from the start but rather to begin with a selected few, ensuring that Insurance CH would meet compliance requirements while laying the groundwork for future expansion.

The project thus clarified the roles related to reference architecture within the organization and

created this initial set of reference security architectures. The initial set of reference architectures was created to address the most common scenarios encountered in the organization's software applications. These templates were inspired and generalized from selected applications at Insurance CH to ensure they were both practical and relevant to the organization's specific needs.

Through the focused development of five Reference Security Architectures, the project successfully met compliance requirements while establishing a foundation for future growth and expansion in secure application development.

## Chapter 2

# Third-Party Library Management

This chapter presents the solutions designed and implemented at Insurance CH to comply with the requirements of Insurance & CO's Information Security Policy on the management of third-party libraries.

In this project, the initial solution for managing third-party libraries was approved by the CISO Team but rejected by the DevOps Team for being too complex and difficult to operate. This led to the creation of a more practical and efficient solution. This chapter discusses the technical and non-technical challenges faced during this process, highlighting the importance of balancing security requirements with operational feasibility.

The remainder of this chapter is organized as follows. First, section 2.1 begins with a presentation of the technical requirements mandated by the Information Security Policy. Section 2.2 details the specific infrastructure and tools we had to work with. Then, section 2.3 outlines the initial solutions and discuss why it was rejected. Finally, 2.4 presents the final solutions.

### 2.1 Requirements

The requirements mandated by the Information Security Policy of Insurance & CO aim to ensure the secure management of third-party libraries. These requirements can be categorized into several key areas:

- Introduce a central repository of dependencies that all software can be built from.
- Ensure that dependencies with known high or critical vulnerabilities cannot be used to build software by blocking these dependencies within the build pipeline.

- Regularly review dependencies to ensure they are correctly licensed, actively supported, maintained, current, and justified for inclusion. Ensure they come from trustworthy sources and do not have known high or critical vulnerabilities.
- Use automated tools to scan for vulnerable dependencies and assign identified issues to the respective development teams.
- Ensure a process is in place to create a Software Bill of Materials (SBOM) for applications, listing all software components and their versions.

These requirements define the technical objective of this projects, that is, create a robust framework for managing third-party libraries, mitigating security risks, and ensuring compliance with legal and licensing standards. The subsequent section will discuss the specific infrastructure and tools available for implementing these requirements.

## 2.2 Current State & Tools

The infrastructure for third party dependency management relies on a suite of tools that are already in place within Insurance CH, each fulfilling a role to make the overall solution functional.

- **JFrog Artifactory:** Serves as a centralized repository for third party dependency management. It not only stores generated artifacts during build processes but will also contain all the third party components needed by the developers.
- **Jenkins:** With its integration alongside Artifactory, Jenkins enables the automation of tasks such as the seamlessly fetching of dependencies and the uploading of additional third party components from the internet within Artifactory.
- **GitHub:** GitHub hosts developers' repositories and configuration files specifying the Artifactory URL. The code that will be committed and run, will be built by Jenkins.
- **JFrog Xray:** JFrog Xray is a security scanning tool that offers comprehensive visibility into the security and compliance of software components. Xray allows the scan of artifacts, containers, and software dependencies for vulnerabilities.
  - Security and Compliance analysis: Detect and prioritize security vulnerabilities and compliance issues within the artifacts.
  - Insecure use of libraries and services: Discover whether common Open Source Software libraries and services are used or configured insecurely, causing exposure to attacks
  - Exposed Secrets: Detect secrets left exposed in any containers stored in JFrog Artifactory.

- Automated SBOM: Create and export SBOM thanks to a comprehensive analysis of both direct and transitive dependencies.
- Integration with CI/CD Pipelines: Seamlessly integrates with tools like Jenkins and Artifactory and existing CI/CD pipeline, facilitating the automation of security checks.
- Reports and Dashboards: Generates comprehensive reports and dashboards, offering insights into vulnerabilities.

## 2.3 Initial Solutions

A gap analysis was conducted, and two main security flaws were identified. First, Artifactory was configured to mirror several major public repositories, providing developers with unrestricted access to a wide range of libraries and plugins. This configuration lacked any controls or restrictions on which components could be accessed or integrated into the production environment. Second, there were no formal processes in place for the addition, or maintenance of these third-party components, resulting in an inability to assess or manage the associated security risks effectively.

The unrestricted mirroring of public repositories, such as Central Maven, which contains more than 14 millions packages with multiple versions [25], exposed the development environment to significant security vulnerabilities. With the absence of proper control, developers could inadvertently introduce vulnerable or outdated libraries and plugins into the software lifecycle. This uncontrolled integration of third-party components posed a substantial risk, making it challenging to ensure that all dependencies were secure, up-to-date, and free from known vulnerabilities.

The lack of a structured process for managing third-party dependencies further compounded these risks. Without clear guidelines or controls, the development environment was vulnerable to incorporating insecure components, potentially increasing the attack surface and compromising the security of the business applications.

To mitigate these identified gaps, a comprehensive approach was necessary. This included designing and implementing the necessary controls to limit access to specific repositories, establishing formal processes for the approval and management of third-party libraries and plugins, and continuously monitoring their security status.

Our initial solution addressed the identified security flaws by transitioning to a centralized local inventory and remove all mirrored repository. This would have allowed granular control over the third-party components and their versions. The aim was to ensure that only approved third-party components were contained within the company's Artifactory.

Developer's access to public repositories through Artifactory was going to be prohibited, and an onboarding process was defined for any additional third-party components required by the



developers. This process included assessing the risks and vulnerabilities of the third-party library or plugin.

With this solution, it was ensured that only authorized and secure dependencies were fetched and utilized by the applications, thereby considerably reducing the attack surface. However, it was not feasible to assess the numerous currently utilized components that have been used over several years and mitigate all their potential vulnerabilities. Therefore, our strategy consists of conducting a risk acceptance for these third-party components after a scan of their vulnerabilities. For critical issues that were identified, another initiative needed to be taken, which would be out of the scope of this transformation.

### 2.3.1 Implementation

To reach the solutions, a four-step implementation was proposed as shown in 2.1.

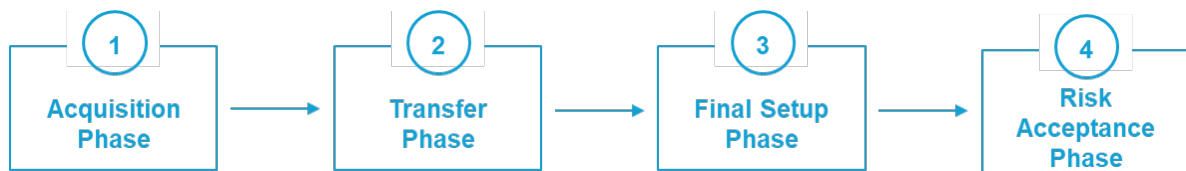


Figure 2.1: Implementation Phases for the Third-Party Management Solution

#### 2.3.1.1 Phase 1

In this first implementation phase, the aim is to know what direct and transitive third-party components are currently being utilized in order to directly include them within a centralized inventory. It begins with configuring remote/mirrored repositories within Artifactory for each technology to ensure that they cache all dependencies.

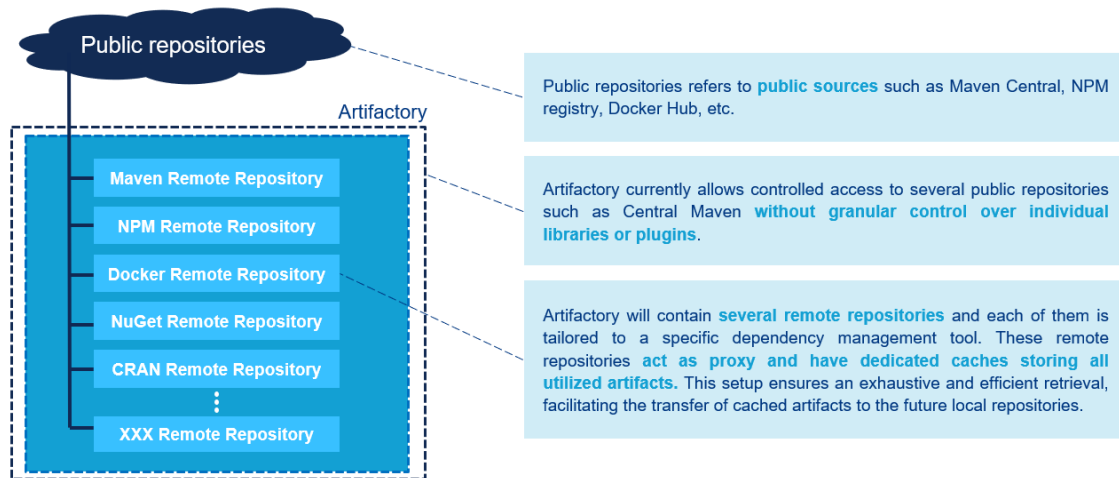


Figure 2.2: Phase 1 of the implementation of the Third-Party Management Solution

#### 2.3.1.2 Phase 2

During the second phase of the implementation, all the third-party components cached by the remote repositories will be transferred to the corresponding local repositories. It aims to ensure that all the necessary dependencies are made available within the centralized inventory.

#### 2.3.1.3 Phase 3

With the completion of the repository transfer phase, the diagram below offers an insight into the finalized setup of the solution and details the technical configurations and operation of the solution.

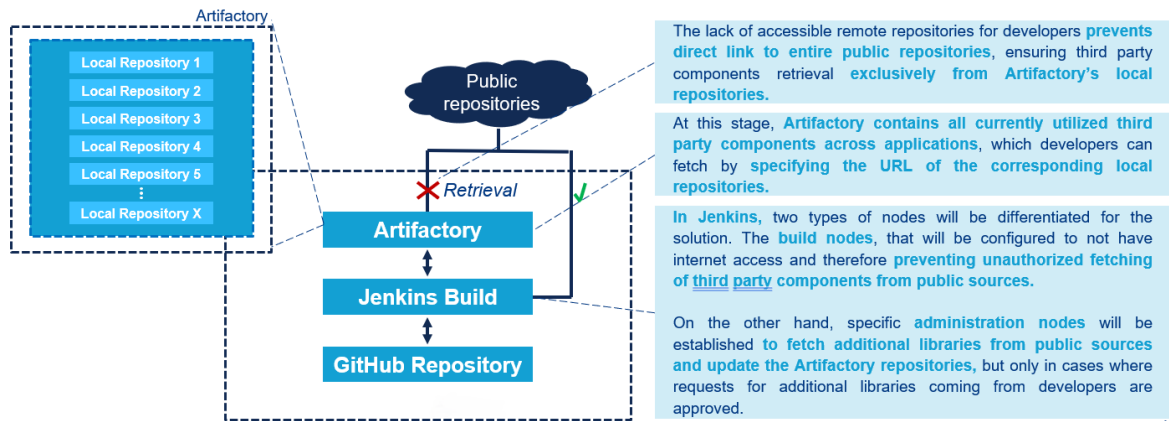


Figure 2.3: Phase 3 of the implementation of the Third-Party Management Solution

#### 2.3.1.4 Phase 4

The preceding implementation steps have provided a comprehensive list of third-party libraries and plugins used by developers. Given the extensive nature of this list, conducting a risk assessment for each component is not possible. Therefore, a risk acceptance must be performed.

To perform the risk acceptance, it is important to have a view of the vulnerabilities in scope. It is possible by leveraging Xray. The scan analysis will give an overview of the vulnerabilities based on criticality, helping in making an informed decision, considering potential project impacts in terms of cost/benefit associated with mitigating vulnerabilities related to all these third-party components.

The decision regarding the first risk acceptance will require approval from a business stakeholder and must be documented for compliance and reference purposes.

#### 2.3.2 Onboarding and Maintenance Processes

The processes are almost identical to those in the final solutions and are presented exclusively there.

### 2.4 Final Solutions

With the previous solution, the DevOps Team raised some concerns such as the complexity of the processes and associated costs, the handling of transitive dependencies, and the future stability of the Artifactory setup. The new solution aims to address these concerns while minimizing potential

drawbacks and maintaining compliance with ISP6. The whitelisting requirement from ISP6 can be fulfilled by defining that a library is whitelisted if it does not have high or critical vulnerabilities.

The new solution is way simpler. It focuses on protecting the production environment with less manual intervention. The Artifactory server remains connected to the Public Repositories on the Internet. The production and development environments are protected by Xray. Xray ensures that the third-party components that are utilized do not contain vulnerabilities with a non-acceptable severity level. Finally to still maintain a level of security, three processes are defined for onboarding, continuous maintenance and annual maintenance.

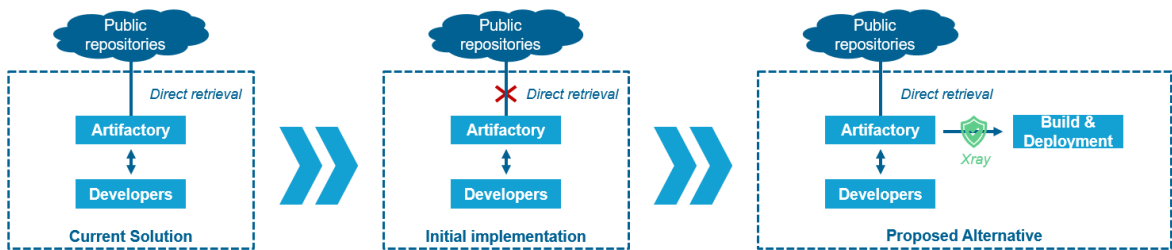


Figure 2.4: Overview of the alternatives

### 2.4.1 Implementation

The new solution revolving around Jfrog Xray, configuration by the DevOps team is required in 3 main steps, visible in Figure 2.5. They respectively consist of analyzing the artifacts in scope and their dependencies with indexation, defining the scope of the monitoring with watches.



Figure 2.5: Implementation Steps for the Final Third-Party Management Solutions

#### 2.4.1.1 Step 1

The first step to configure Jfrog Xray is to index the repositories. Indexing repositories that contain build artifacts enable Xray to scan and analyze these artifacts for vulnerabilities

The screenshot shows the 'Indexed Resources' page in Xray. It has tabs for 'Repositories (64)', 'Builds (0)', and 'Release Bundles'. The 'Repositories' tab is active, showing a table of 64 repositories. A callout '1' points to the 'Add a Repository' button. Another callout '2' points to the 'Index Now' button. A third callout points to a warning icon with the text: 'Configuring Xray requires prior integration with Jenkins to enable build failures.'

Repository Key	Type	Package Type	Index Status	Retention
local-docker-releases	Local	Docker	14 / 1054 (1%)	Forever
local-docker-snapshots	Local	Docker	21 / 538 (3%)	Forever
local-docker-training	Local	Docker	0 / 0 (0%)	Forever
local-generic	Local	Generic	1 / 39 (2%)	Forever
local-generic-releases	Local	Generic	7 / 150 (4%)	Forever
local-generic-snapshots	Local	Generic	0 / 0 (0%)	Forever
local-m2-3rd_party	Local	Maven	1 / 832 (0%)	Forever
local-m2-attlassian-plugin-sdk-8.0.16	Local	Maven	0 / 157 (0%)	Forever
local-m2-releases	Local	Maven	20 / 20 (100%)	Forever
local-m2-snapshots	Local	Maven	0 / 0 (0%)	Forever

**Callout 1:** Add a Repository

**Callout 2:** Index Now

**Warning:** Configuring Xray requires prior integration with Jenkins to enable build failures.

Figure 2.6: Step 1 of the Final Third-Party Management Solutions

### 2.4.1.2 Step 2

Watches allow to specify the scope of what is going to be monitored by Xray, ensuring that all relevant artifacts are continuously scanned for vulnerabilities. A watch per team will be created to ensure that the respective application teams are notified whenever high or critical vulnerabilities are raised.

For each watch, a group mailing list will be designated as the watch recipient. The configuration and control of this mailing list will be delegated to the respective application team, allowing to update the list as necessary and ensuring that notifications reach the appropriate members in case of team changes.

All the repositories falling within the scope of the watch shall be included. Since a repository may contain build artifacts from multiple application teams, a regular expression-based filter can be defined for each repository. For example all artifacts whose path contains "TEAM\_NAME" can be scanned.

### 2.4.1.3 Step 3

Policies define the criteria and actions that should be taken when vulnerabilities are detected, allowing to automate response and ensure consistent handling of issues. The build policy blocks builds with high or critical vulnerabilities, preventing insecure deployments. The runtime policy monitors deployed environments for new vulnerabilities in current libraries, ensuring continuous security.

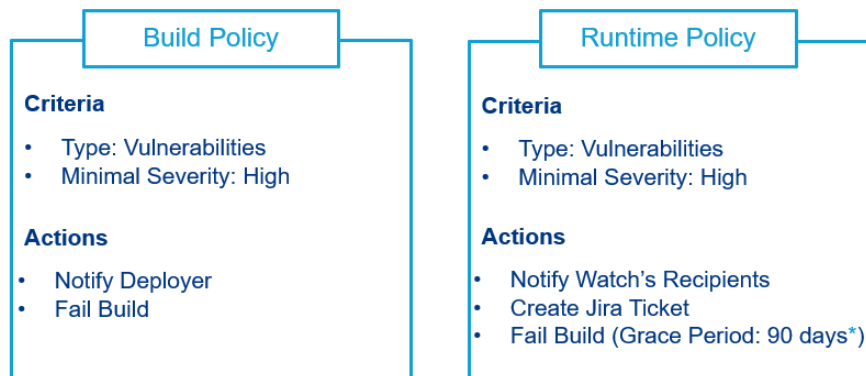


Figure 2.7: XRay Policies in the Final Solution

## 2.4.2 Process: Onboarding New Third-Party Dependencies

Any future additional library and plugin requested by developers will undergo analysis for approval with a defined onboarding process as visible in 2.8. The onboarding process ensures that new third-party components are securely integrated into the development environment. It guarantees that code cannot be built if high or critical vulnerabilities are detected and addresses cases where using a third-party component is necessary despite its vulnerabilities.

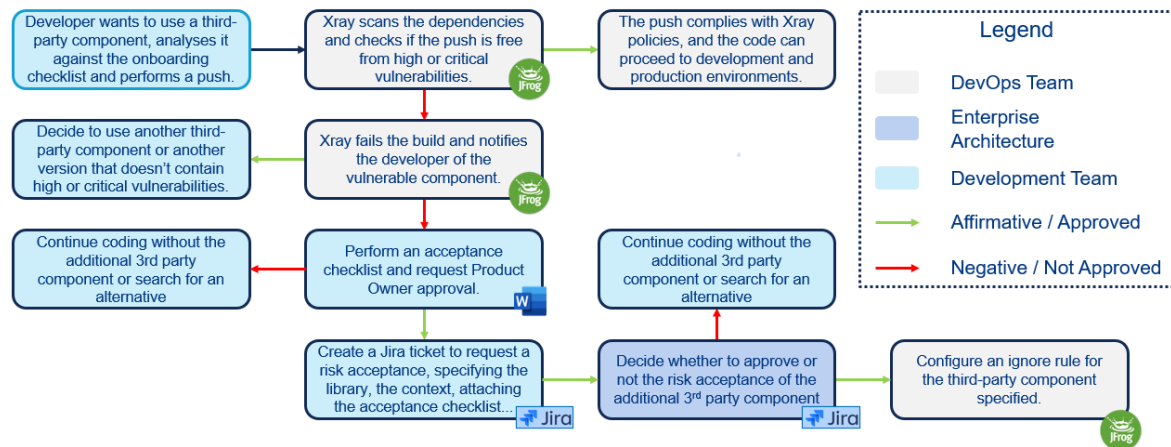


Figure 2.8: Onboarding process for new third-party dependencies

### 2.4.3 Process: Continuous Maintenance

Acknowledging that the centralized inventory can become a risk over time due to the continuous addition of new components and the discovery of new vulnerabilities, it is essential to maintain a continuous maintenance process. Without regular oversight, issues can quickly pile up, compounding the risk. Therefore, security champions need to address new vulnerabilities as they arise, aiming to ensure that all utilized components remain secure. The process is visible in Figure 2.9.

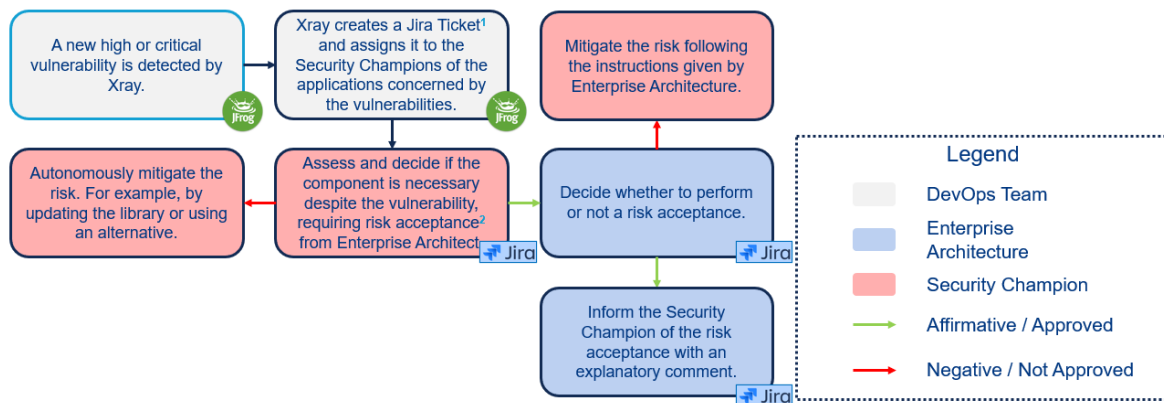


Figure 2.9: Continuous process for third-party dependencies

### 2.4.4 Process: Annual Maintenance

It is crucial that Enterprise Architecture remains aware of the risks associated with third party components across all the applications. Therefore, an annual clean-up and risk assessment must be conducted regarding the third-party components, in addition to exporting and documenting a Software Bill Of Materials (SBOM) per application. (product owner approval, checklist). The process is visible in Figure 2.10.

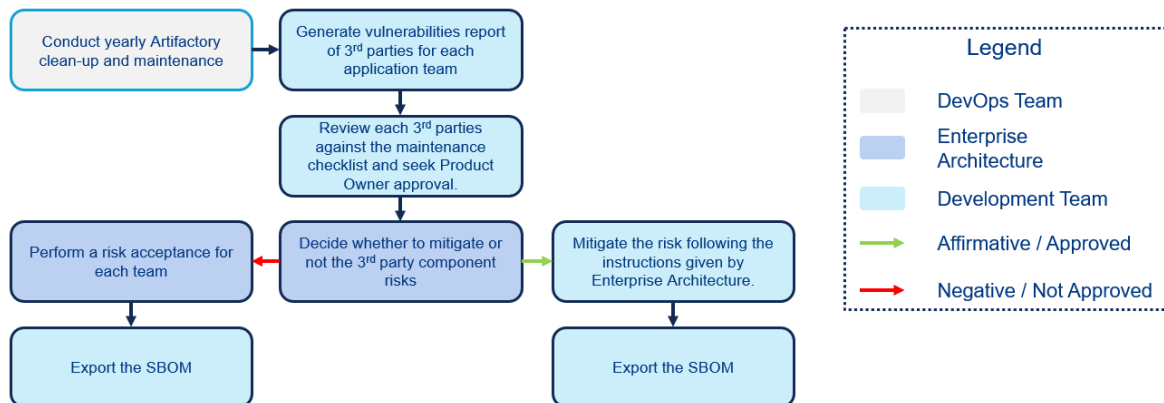


Figure 2.10: Annual process for third-party dependencies

## 2.4.5 Drawbacks and Benefits

Table 2.1 highlights the benefits and drawbacks of this new solution. The benefits show that the solution is indeed simpler, but the drawbacks highlight that it decreases security.

Benefits	Drawbacks
<ul style="list-style-type: none"> <li>• Simpler onboarding process</li> <li>• No restructuring work within Artifactory</li> <li>• Initial risk acceptance facilitated</li> <li>• Better response time to vulnerabilities</li> <li>• Easier handling of transitive dependencies</li> </ul>	<ul style="list-style-type: none"> <li>• The onboarding process only applies to vulnerabilities violating the defined policies, and the vulnerabilities that are not under the scope of the policies or within the Xray database are not detected.</li> <li>• The risks brought by vulnerabilities out of the scope of the watches won't undergo a risk acceptance by Enterprise Architecture.</li> </ul>

Table 2.1: Benefits and Drawbacks



## Chapter 3

# Secret Management

This chapter presents the solution implemented at Insurance Technology to comply with the requirements of Insurance & CO's Information Security Policy on Secret Management. The objective was to establish a secure and standardized method for handling passwords, API keys, encryption keys, and more, within applications. The solution aimed to close existing security and compliance gaps while creating a scalable framework for future needs.

Previously, a Proof of Concept (PoC) for HashiCorp Vault had been developed but remained unused due to the lack of a defined target operating model. Without a clear framework or ownership, the PoC was not adopted by development teams, most of whom were unaware of its existence. Moreover, no authentication mechanisms or processes were set up to give access to developers. This led to developers hardcoding secrets directly into applications, posing significant security risks and resulting in non-compliance with the Information Security Policy.

To resolve these issues, a comprehensive solution was designed and implemented using HashiCorp Vault. A namespace structure was established to provide logical separation for various environments and development teams, and Role-Based Access Control (RBAC) was integrated with Azure Active Directory to manage secure access. OIDC authentication was configured to ensure secure user access. Additionally, detailed processes were defined for managing the lifecycle of secrets, and the solution was integrated with Kubernetes clusters and Jenkins pipelines to automate secret handling during application deployments, thereby ensuring a robust, secure, and compliant framework for managing secrets.

The rest of this chapter is organized into two sections. The first section outlines the requirements set by Insurance & CO's Information Security Policy for secret management. The second section details the designed and implemented solution.

## 3.1 Requirements

Insurance & CO's Information Security Policy outlines requirements for managing secrets. These requirements are standard and align with industry practices to ensure sensitive secrets are protected and data is secure. Key aspects of the policy include:

- **Secure Storage:** Secrets must be stored in secure environments like vaults, not in code repositories or configuration files, to prevent accidental exposure.
- **Environment Segregation:** Production secrets should be strictly separated from development and testing environments, which typically have weaker security controls.
- **Access Control and Encryption:** Secrets must be encrypted both in transit and at rest, and access should be restricted to authorized personnel only. This minimizes the risk of data breaches.
- **Dynamic Provisioning:** Secrets should be provided dynamically to applications or containers only when needed, ensuring they are not statically embedded in code or configurations.
- **Continuous Monitoring:** Automated tools should be used to regularly check for secrets in code repositories and ensure they are removed if found.

## 3.2 Solution

### 3.2.1 Overview

On a high level, HashiCorp Vault is a service where you can securely store and manage secrets. It operates as a client-server application where the server is a single, secure location for secrets, and clients access and manage those secrets. Figure 3.1 illustrates the operation of HashiCorp Vault along with the interactions that occur between the Security Champion, the Vault, and applications. A Security Champion is a designated member of a development team responsible for promoting and ensuring the implementation of security best practices throughout the development process.

- The security champion stores different types of secrets within the vault, such as API keys and database (DB) root certificates, rather than embedding them directly into the code. This ensures an enhanced level of security and minimizes the risks of exposure or compromise of sensitive information.
- Applications interact with the Vault to securely access the secrets required for their operations. As shown in Figure 3.1, one application may read the API key and/or the root certificate

of a database from the Vault, ensuring secure access to the necessary credentials and/or cryptographic identities required for establishing secure connections and communications with other services.

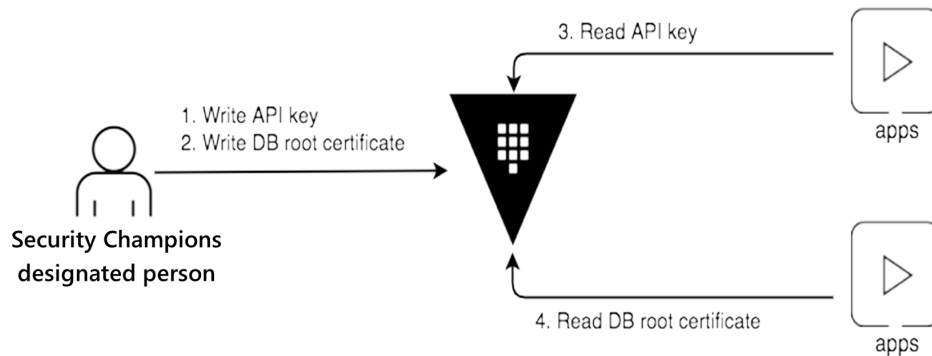


Figure 3.1: Overview of HashiCorp Vault Operation and Interactions

### 3.2.2 Separation in Namespaces

In HashiCorp Vault, a namespace serves as a logical grouping mechanism that allows organizations to partition and isolate data, policies, login paths, and configurations within a single Vault instance. Namespaces provide a way to create distinct and self-contained environments within the Vault, enabling development teams to manage their own secrets and access policies independently while sharing the same underlying infrastructure.

Each namespace has its own:

- Policies
- Authentication Methods
- Secrets Engines (see Section 3.2.5.1)
- Tokens
- Identity entities and groups

The secrets are segregated based on development teams and environments. Each application or team, referred to as a “domain,” is given three namespaces, one for each maturity level:

- **Eng:** The engineering environment is for developers and engineers to write and test their code, allowing them to experiment, debug, and validate their work before moving to more formal testing stages.

- **Non-Prod:** The non-production environment is where most of the software development cycle occurs, from initial development to various phases of testing.
- **Prod:** Production is the environment where the application is fully operational and accessible by the end users.

The namespaces adhere to the following naming convention: `inch/<maturity>/<domain>`.

For instance, the application team ABC will have the following namespaces:

- `inch/eng/abc`
- `inch/non-prod/abc`
- `inch/prod/abc`

### 3.2.3 Authentication for Users

#### 3.2.3.1 Access Rights

To provide access permissions for users to the HashiCorp Vault secret structure and the secrets themselves, Azure AD (AAD) dedicated security groups for RBAC management are available and used within Insurance CH. AAD is a cloud-based directory service and serves as an identity and access management service, allowing for centrally managed authentication, authorization, and user account management.

The groups are available for each namespace with the following structure:

`inch-HashiCorp-<maturity>-<domain>-<permissions>`

For example, for read permissions on the non-prod namespace of the “abc” application, the following AAD security group is utilized: `inch-HashiCorp-non-prod-abc-read`.

The groups are managed by the group owners from the Cloud team and/or HashiCorp security team. Before implementing membership changes, approval is required from one of the group owners and from the Chapter Lead of the new group member to add.

To request specific access permissions to particular namespaces, developers have to:

1. Determine the needed security group.

2. Submit the access request to the corresponding group owner and wait for its approval, in addition to the approval from the Chapter Lead.
3. Wait for the processing of the request once it has been approved.

Once access has been granted, a notification is sent, and access to the secrets and namespaces related to the security group is provided.


### 3.2.3.2 Authentication Process

For developers with the correct access rights, authentication to the Vault service is made through the HashiCorp Vault user interface accessible via a web browser.

To authenticate, navigate to the Web UI.

- In the **Namespace** box, type in the application namespace.
- For the authentication method, select **OIDC**.
- Finally, type in the role. Leaving this box blank means signing in with the default role, if one is configured.
- Click the **Sign In** button to authenticate.

Refer to Figure 3.2 for a visual representation of the process.

### Sign in to Vault

Namespace

azchy/non-prod/abc

Method

OIDC

Role

role-devops

Leave blank to sign in with the default role if one is configured

More options

Sign in with OIDC Provider

Figure 3.2: Authentication Process for HashiCorp Vault

Upon logging in, the authentication request is forwarded to AAD, granting access to HashiCorp Vault. After logging in to a domain, it is possible to add, update, and remove secrets from the Vault.

### 3.2.4 Authentication for Applications

Secret access is only possible with applications deployed on a registered cluster. To assure seamless communication between our Kubernetes clusters and Vault, each cluster needs to be registered using its unique certificate and token. However, this process has been automated by the DevOps team.

For access within pods, the Vault Container Storage Interface (CSI) provider is used. It allows the Kubernetes pods to consume Vault secrets using ephemeral CSI secrets store volumes. This means that the secrets will be mounted as file in the pod's filesystem.

### 3.2.5 Adding Secrets

#### 3.2.5.1 Secrets Engines

Secrets must be added to HashiCorp Vault during the development of applications, as this enables secure management and access to sensitive information. DevOps facilitates this process by offering

secret engines within HashiCorp Vault, allowing security champions to add secrets directly into the Vault.

Secrets engines are specialized components designed for specific types of secrets or use cases. A secret engine is a virtual filesystem mounted at a unique and specific path within Vault. This path determines where in Vault's namespace the engine will operate and acts like a router listening for requests related to the secrets it manages. Vault then directs secret requests to the appropriate secret engine based on the path prefix specified in the request.

There are several types of secret engines that cater to different needs, such as:

- **AWS Secrets Engine:** Dynamically creates AWS access keys for a specified duration and is utilized by applications that need to interact with AWS services by providing them with temporary credentials.
- **Transit Secret Engine:** Provides encryption as a service, allowing data to be sent to Vault for encryption without storing it in Vault.
- **Cubbyhole Secrets Engine:** Each token in Vault has its own isolated storage area (Cubbyhole) where secrets are stored securely, and only the token that created the cubbyhole can access it. It is utilized when secrets need to be accessible only by the token holder.

#### 3.2.5.2 Key-Value Secrets Engine

For now, the secret engine utilized within Insurance CH is the Key-Value (KV) engine. It stores secrets within the Vault as key-value pairs, allowing access to anyone with the appropriate permissions set within Vault. This secret engine can operate in two modes:

- **KV1:** Provides basic storage for key-value pairs with simple CRUD (Create, Read, Update, Delete) operations.
- **KV2:** Extends KV1 by adding versioning capabilities to the secrets, allowing different versions for each key that is stored.

To create a secret within a Vault, the following steps are to be followed:

1. From the Dashboard, navigate to the desired section.
2. Click on the secret engine in which the secret must be added.
3. Click the **Create Secret** button.

4. Fill in the secret's path where it must be stored. In the example illustrated in Figure 3.3, if the folders `eng` and `apikey` currently don't exist, they will be automatically created.
5. Fill in the blue section of Figure 3.3 with the desired name for the secret.
6. Fill in the green section of Figure 3.3 with the secret itself.
7. Click the **Save** button to create the secret.

< kv-v1

### Create Secret [Delete secret >](#)

☐ JSON

Path for this secret

eng/apikey/Google

Secret Data

key	AAaaBBccDDeeOTXzSMT1234BB_Z8JzG7JkSVxl	<a href="#">Add</a>
-----	--	---------------------

[Save](#) [Cancel](#)

Figure 3.3: Creating a Secret in HashiCorp Vault



## **Chapter 4**

# **Threat Modeling and Reference Security Architectures**

In this chapter, we explore the solutions implemented to fulfill the requirements of the Information Security Policy on Threat Modeling and Reference Security Architectures. Despite their distinct sets of requirements and unique characteristics, these subjects are addressed together due to their shared nature and challenges. Both areas faced significant difficulties in terms of time and cost, necessitating a cost-effective and time-efficient approach to achieve compliance.

The subsequent sections of this chapter are organized as follows. There are two main sections: Threat Modeling and Reference Security Architectures. Each section is structured similarly. First, a bit of theoretical background is given on the subjects, then the specific requirements are outlined, followed by a description of the solutions implemented to meet these requirements effectively and cost-efficiently.

### **4.1 Threat Modeling**

#### **4.1.1 Theoretical Background**

Threat modeling is a proactive and systematic approach employed to identify, assess, and mitigate potential security threats within a software system [26]. Threat modeling aims to foresee potential security issues early in the software development lifecycle, allowing for the design and implementation of effective countermeasures before the threats can materialize into vulnerabilities. By systematically analyzing a system's architecture, design, and implementation, threat modeling helps to identify areas of potential risk and prioritize them based on their impact and likelihood.

The primary objectives of threat modeling include [27]:

1. Enhancing security posture: By identifying and addressing potential threats early in the software development lifecycle, threat modeling helps to build more secure systems, reducing the risk of security breaches.
2. Protecting sensitive data: Ensuring the confidentiality, integrity, and availability of sensitive information by identifying vulnerabilities that could lead to data exposure or corruption.
3. Reducing costs: Preventing security issues early in the development process minimizes the costs associated with fixing vulnerabilities later, which can be significantly higher.
4. Complying with regulations: Meeting industry standards and regulatory requirements for security.
5. Improving stakeholder confidence: Demonstrating a commitment to security enhances trust and confidence among stakeholders.

#### **4.1.2 Requirements**

The requirements of the Information Security Policy on Threat Modeling align with the previous definition and objectives of threat modeling but give more specific details on how this should be achieved. They require the creation of a documented threat model for applications. The threat model must use a standardized methodology such as STRIDE and include diagramming, threat identification, exploitability checks, design flaw mitigation, and validation of threat model artifacts.

The next section will detail the solutions implemented to meet these requirements effectively and cost-efficiently.

#### **4.1.3 Solution**

This section outlines the methodology used for threat modeling. The proposed methodology is designed to integrate with the existing Overarching Risk Assessments.

Section 4.1.3.1 details the integration of the methodology with the Overarching Risk Assessment. Section 4.1.3.2 elaborates on the overall methodology and explains the building blocks used in the methodology itself. Section 4.1.3.3 clarifies the scope and security assumptions. Finally, Section 4.1.3.4 explains practically how to use the methodology and Section 4.1.3.5 gives the models.

#### **4.1.3.1 Integration with the Overarching Risk Assessment**

To effectively implement threat modeling, it is essential to integrate this new process with the existing Overarching Risk Assessment. This integration ensures that threat modeling is not an isolated process but is aligned with the broader risk management efforts.

The Overarching Risk Assessment evaluates the inherent risk associated with an application to propose relevant security measures. The Overarching Risk Assessment must be completed by each application when the application is initiated and on a yearly basis. It is composed of two documents. The first one is a questionnaire referred to as IS\_Schutzbedarfsermittlung, as shown in Figure 1. The IS\_Schutzbedarfsermittlung is an instrument to understand the assets being protected, their criticality, the potential damage of data breaches, and whether regulatory requirements are applicable. The ensemble of the answers to these questions determines the inherent risk of an application. Based on the resulting inherent risk, the IS\_Risikobewertung, as shown in Figure 2, determines the set of applicable security measures to be implemented. These security measures include a wide range of protective actions such as implementing encryption to protect data, enforcing authentication to verify user identities, setting up authorization to control access levels, and enabling logging to monitor and track system activities.

However, the Overarching Risk Assessment lacks a clear link between the questionnaire results and the specific security measures that need to be implemented. More precisely, it is unclear what are the exact threats mitigate by the applicable measures. This is where threat modeling comes in. The proposed threat modeling methodology serves as a bridge between the inherent risk given by the IS\_Schutzbedarfsermittlung and the applicable security measures provided by the IS\_Risikobewertung. By doing so, it provides developers with a clear understanding of possible threats to their applications and the corresponding security measures required to mitigate these threats.

Schutzbedarfsermittlung	
1	Wie sind die sensiblen Daten klassiert, welche mit diesem System generiert, verarbeitet, transferiert und / oder gespeichert werden? (Verwenden Sie zur Bewertung bitte die <a href="#">Datensciz-Tabella</a> im Anhang).
Bitte wählen:	
2	Wie hoch schätzen Sie den Schaden bei Verlust der Vertraulichkeit der Daten? (Verwenden Sie zur Bewertung bitte die <a href="#">Datensciz-Tabella</a> im Anhang).
<div> <div>Finanziell</div> <div>Reputation</div> <div>Regulatorisch</div> <div>Operativ</div> </div>	
Bitte wählen:	
3	Unterliegen Daten, welche durch das System verarbeitet werden, regulatorischen Vorgaben (z.B.: Geschäftsgeheimnisverordnung, Bundesgesetz über den Datenschutz)?
Bitte wählen:	
4	Wer hat Zugriff auf das System und / oder Daten? (Nennen Sie alle auftretenden Akteure)
<div> <input type="checkbox"/> Kunden         <input type="checkbox"/> Mitarbeiter         <input type="checkbox"/> Partner / Lieferanten / andere externe         <input checked="" type="checkbox"/> Öffentlichkeit       </div>	
Bitte wählen:	
5	Werden Daten der Allianz Suisse ausserhalb der Schweiz übertragen, verarbeitet, gespeichert?
Bitte wählen:	

Figure 4.1: Caption for image 1

Risikobewertung / Umsetzung Schutzmassnahmen	
ID	Schutzmassnahme / Anforderung
00-Premio-01	Die Allianz Technology SE hat ein gültiges und vollständiges "Asset and Risk Assurance Questionnaire" (ARAQ) sowie einen dazu passenden "Asset Assurance Plan" (AAP) zur Verfügung gestellt, dessen Massnahmen durch das Projekt oder Vorhaben umgesetzt werden.
00-Premio-02	Der Dienstleister bestätigt, dass seine interne Organisation sowie seine nachgelagerten Dienstleister, die geltenden rechtlichen Anforderungen und Vorschriften, einschliesslich der EU-DSGVO, einhalten.
00-Premio-03	Der Dienstleister bestätigt, dass seine interne Organisation sowie seine nachgelagerten Dienstleister, die geltenden rechtlichen Anforderungen des Schweizer Datenschutzgesetzes (DSG) einhalten.
00-Premio-04	Bedürfnisse des Umgangs mit personenbezogenen Daten wurden jegliche relevante Anforderungen gemeinsam mit dem internen Datenschutzbeauftragten identifiziert (PIA: Privacy Impact Assessment). Entsprechende datenschutzrechtliche Einschätzungen sind dokumentiert, allfällige Massnahmen nachgefragt und der/die Datenschutzbeauftragte stimmt der Nutzung der Anwendung zu.
00-Premio-05	Bei der Beschaffung von Diensten Dritter ist zu prüfen, ob mittels eines Service Level Agreements (SLA) die Anforderungen an die Sicherheit, Verfügbarkeit und Qualität der Lösungen festzulegen sind.
00-Premio-06	Die Architektur der Applikation bzw. des IT Services liegt dokumentiert vor und ist vom geeigneten Architektur Board und vom Business Owner bzw. IT Service Owner genehmigt. Die Dokumentation umfasst Komponenten, Netzwerkarchitektur inkl. URLs und deren IP-Adressen, Schnittstellen, Batch-Jobs, Datenflüsse und Datenstrukturen, einschliesslich deren Klassifizierung, sowie was wo verschlüsselt ist.
00-Premio-07	Die Architektur erfüllt die Verfügbarkeitsanforderungen, auch hinsichtlich maximal zulässigem Datenverlust im Störfall (RPO: Restore Point Objective), z.B. durch Nutzung geeigneter Elemente wie Redundanzen, Datensicherungen, Daten-Synchronisation/Spiegelung, geeignete Betriebsorganisation etc.
00-Premio-08	Der Application Owner und eine angemessene Betriebs- und Supportorganisation sind definiert.
00-Premio-09	Der Business Owner hat mit dem BCM-Officer geprüft, ob die Applikation in den BCM-Katalog aufgenommen ist und dies auf <a href="#">www.bcm.ch</a>

Figure 4.2: Caption for image 2

#### 4.1.3.2 Overview of the Methodology

The threat modeling methodology involves visualizing an application as simple architecture building blocks. This approach allows each architecture building block to be threat-modeled separately. Developers select the relevant building blocks applicable to their specific application. By combining these blocks, developers create a comprehensive threat model tailored to their particular application.

The threat model for each architecture building block encompasses the following components:

1. **Scenario:** Each building block is designed to analyze a particular part of an application. This ensures that the threat model is both relevant and precise, focusing on the specific security needs and potential vulnerabilities associated with that scenario.
2. **Technical Architecture Diagram:** Each building block includes a detailed visualization of the system components and their interactions, aiding in the identification of potential threats. This visualization contains:
  - **The components:** The individual parts of the system that interact with each other, depicted as blue boxes with a name in white.
  - **Communication flows:** The communication flows between components, shown as blue arrows, which illustrate the flow of data and the relationships between different parts of the system. A name is associated to the arrow illustrating the relationship and/or type of data exchanged.

- **Assets:** The assets that need protection, visible in the table on the right and as grey tags positioned on the components where they are stored, used, or processed.
- **Threat Actors:** The potential adversaries that could exploit vulnerabilities within the system. They are represented as components with a red tag on the upper left corner.
- **Attack Paths:** The possible routes through which a threat actor could exploit vulnerabilities to compromise the system. They are represented as dotted yellow arrows.
- **Counter Measures:** The security measures implemented to mitigate the identified threats. They are represented as green tags on the components that implement them.
- **Trust boundaries:** The demarcations within the system that separate different levels of trust, indicating where potential security controls should be applied.

In its simplest form, a technical architecture diagram will look like depicted in Figure 3.

3. **Threat Table:** Each building block contains a table of example threats.

- The table is composed of three columns: the category of the threat, the scenario, and the measures from the Overarching Risk Assessment.
- Each table is made of six rows, one for each category in the STRIDE methodology (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege).

Threat Tables will look like depicted in Table 4.1.

Category	Example of Attack Scenarios	Measures
S	An example threat of Spoofing	Measure 1 Measure 2
T	An example threat of Tampering	Measure 1 Measure 2
R	An example threat of Repudiation	Measure 1 Measure 2
I	An example threat of Information Disclosure	Measure 1 Measure 2
D	An example threat of Denial of Service	Measure 1 Measure 2
E	An example threat of Elevation of Privilege	Measure 1 Measure 2

Table 4.1: Example Threat Table

#### 4.1.3.3 Security Considerations and Responsibilities

The threat modeling methodology outlined in section 3.2 is specifically designed for business applications. Consequently, it only addresses threats managed by the application development teams, excluding those overseen by the DevOps and Cloud teams. This section delineates the respective responsibilities of these teams to clarify the scope and security assumptions of the methodology. First, responsibilities in the production environment, both on-premises and in the cloud, are outlined. Second, those within the CI/CD pipeline are detailed.

In the production environment, application developers are responsible for the security of their application code. This includes implementing secure coding practices, managing dependencies, conducting code reviews, and integrating application-level security measures such as authentication, authorization, and data validation. While developers concentrate on business applications, the DevOps and Cloud teams manage the security of the underlying systems, with their responsibilities varying based on the environment. In on-premises environments, DevOps teams handle the installation and configuration of the entire stack, from the operating system to local Kubernetes clusters. In cloud environments, security responsibilities are shared between the cloud service provider (CSP) and the Cloud and DevOps teams. CSPs like Azure and Red Hat secure the infrastructure and provide running Kubernetes clusters (AKS for Azure and OpenShift for Red Hat), while the Cloud and DevOps teams are responsible for the configurations. Regardless of the environment, the final artifact produced by application developers is a set of containers encapsulating the application. Developers are not concerned by the underlying systems where it will be deployed. Therefore, the threat modeling methodology assumes that these underlying systems are appropriately secured.

In the CI/CD pipeline, developers are responsible for writing unit tests and integration tests to ensure the software functions correctly and securely. On the other hand, the DevOps team, is responsible for maintaining the integrity and security of the CI/CD pipeline. This involves configuring and managing tools like Jenkins, Jira, GitHub Enterprise, and JFrog Artifactory. Additionally, the DevOps team protects against developer errors by establishing a secure build environment and implementing robust user permissions to control access to the CI/CD servers. Developers focus solely on delivering secure and functional code, trusting that the CI/CD infrastructure is properly managed and protected by the DevOps team. In the methodology, all the components managed by the DevOps team are thus out of scope.

In conclusion, the threat modeling methodology focuses on threats managed by application development teams and excludes those handled by DevOps and Cloud teams. This means, that only application-level threats for the business applications are considered in the methodology. On-premises systems, cloud components and CI/CD components are out of scope.

#### 4.1.3.4 Practical Usage of the Threat Methodology

The following steps explain how to perform threat modeling using this methodology in practice.

##### **Step 1: Selection of Appropriate Building Blocks:**

1. Identify relevant use cases:
  - Review the suite of building blocks.
  - Select the building blocks that correspond to the specific features and scenarios of your application.
2. Verify applicability:
  - Ensure that the assets defined in the selected building blocks align with the application's assets.
  - Ensure that the components defined in the selected building blocks align with the application's architecture.

##### **Step 2: Tailoring the Building Blocks to the Application:**

1. Aggregate the selected blocks: Collect the selected building blocks to get a detailed view of the application's architecture with a set of specific threats.<sup>1</sup>
2. Review the selected blocks: Examine and understand the building blocks to see the application from a security perspective.

##### **Step 3: Identification of Threats and Implementing Security Measures:**

1. Read and understand the threats: The collection of all the building blocks already includes relevant threats. This eliminates the need for developers to necessarily search for new threats.
2. Implement security measures: Make sure the countermeasures are correctly implemented and fully mitigate the threats.

#### 4.1.3.5 Suite of Architecture Building Blocks

This section provides the architecture building blocks used in the threat modeling methodology described in Section 3.2. These building blocks are designed to help developers identify and address potential security threats in common architectural patterns.

---

<sup>1</sup>Note that no additional steps are required from developers during this aggregation process. It is simply the process of collecting and presenting all relevant threat models together that gives a comprehensive threat model.

**4.1.3.5.1 Authentication** The Authentication Building Block identifies threats and measures for the authentication process of an external user to an application. The technical architectural diagram in Figure 4.3 illustrates the relevant components and interactions: the user, web service, and authentication server, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

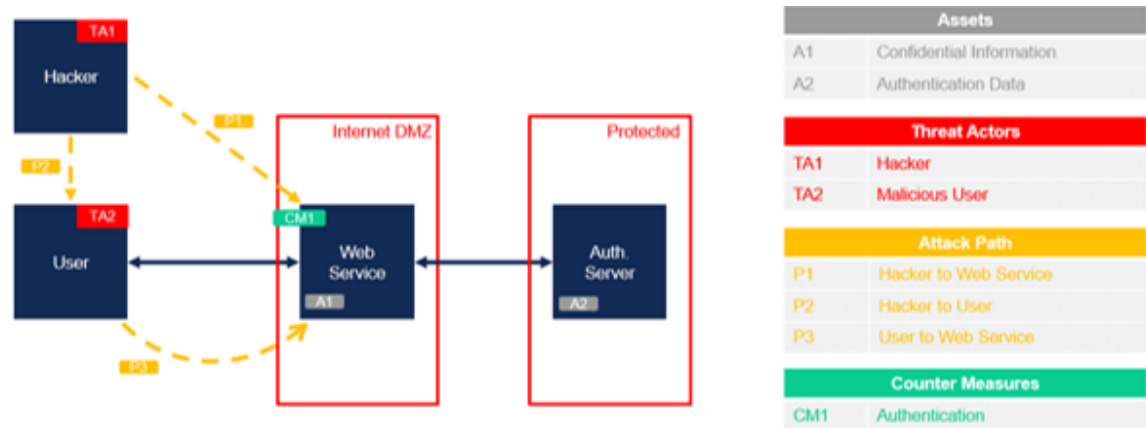


Figure 4.3: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related to Secure Authentication such as Multi-Factor-Authentication and password policies. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.2.



STRIDE	Example of Attack Scenarios	Measures
S	A user impersonating another user using weak authentication	02-Auth-01, 02-Auth-03, 02-Auth-04
T	An attacker using leaked credential to modify the personal information of a user	02-Auth-04
R	A user denying having authenticated or carried out actions post-authentication	02-Auth-16
I	An attacker brute forcing the password of a user to disclose the personal information of that user	02-Auth-14, 02-Auth-19, 02-Auth-28
D	An attacker performing many authentication requests to the Web Service to overload the Authentication Server	05-Perimeter-01, 05-Perimeter-02
E	An attacker exploiting flaws in the authentication process to escalate privileges without authenticating	02-Auth-01

Table 4.2: Threat Table of the Authentication Building Block

**4.1.3.5.2 Certificate Chain of Trust** The Certificate Chain of Trust Building Block identifies threats and measures for the certificate verification process. The technical architectural diagram in Figure 4.4 illustrates the relevant components and interactions: the applications, the third-party presenting the certificate, the certificate authority chain, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

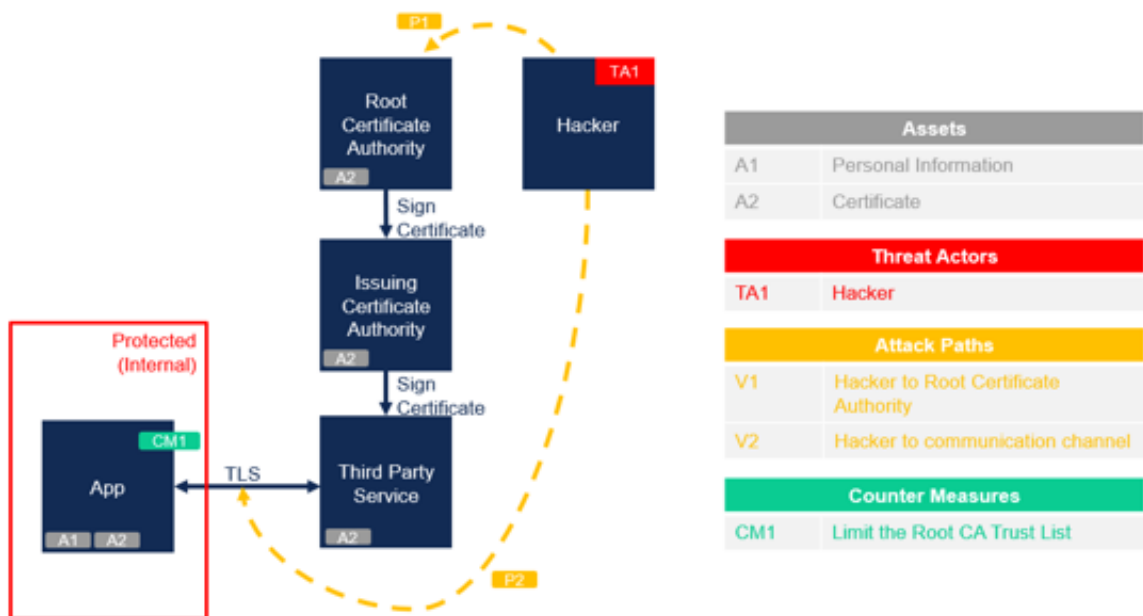


Figure 4.4: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example of attacks scenarios are identified. Those scenarios are mitigated by trusting only a specific set of Certificate Authority. The threats with the ID of the measure from the IS\_Risikobewertung are shown in Table 4.3.

<b>STRIDE</b>	<b>Example of Attack Scenarios</b>	<b>Measures</b>
S	If a trusted certificate authority is compromised or untrustworthy, it could issue certificates allowing to impersonate any website	01-Encrypt-05
T	If a trusted certificate authority is compromised or untrustworthy, it could issue certificates allowing the attacker to alter any traffic between the application and the third-party	01-Encrypt-05
R	N/A	N/A
I	If a trusted certificate authority is compromised or untrustworthy, it could issue certificates allowing the attacker to read any traffic between the application and the third-party	01-Encrypt-05
D	N/A	N/A
E	N/A	N/A

Table 4.3: Threat Table of the Certificate Chain of Trust Building Block

**4.1.3.5.3 CI/CD** The CI/CD Building Block identifies threats and measures for the Continuous Integration and Continuous Deployment processes from the point of view of a developer. The technical architectural diagram in Figure 4.5 illustrates the relevant components and interactions: the developer, the CI/CD servers, the production and staging servers, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

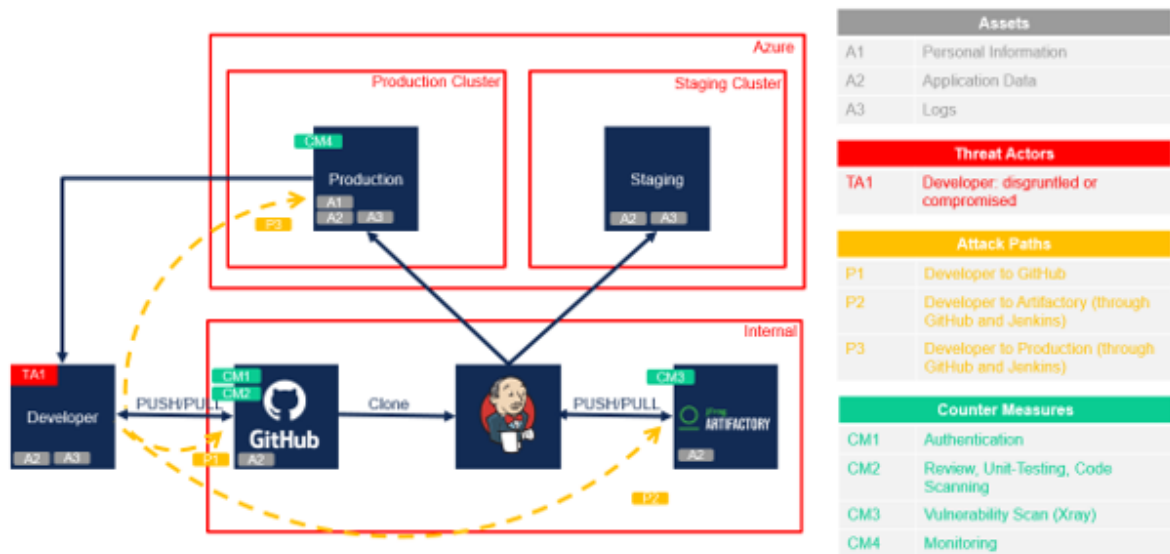


Figure 4.5: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related authentication, input validation and logging. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.4.

STRIDE	Example of Attack Scenarios	Measures
S	An attacker could impersonate a developer Example: stolen credentials, weak authentication	02-Auth-01, 02-Auth-20, 02-Auth-21
T	A compromised developer account could be used to push malicious code to the GitHub repository and thus to production	08-Validation-01, 08-Validation-02
R	A disgruntled developer might deny malicious actions taken, such as pushing harmful code	02-Auth-01, 04-Log-01, 04-Log-02
I	Sensitive information, such as credentials or API keys, could be exposed by a compromised developer account	02-Auth-01, 02-Auth-02, 02-Auth-03
D	Compromised developers accounts could be used to trigger operations that overwhelm the CI/CD infrastructure	No measures in the IS_Schutzbedarfsermittlung
E	Injecting vulnerability in the code could allow an attacker to gain elevated access to production systems and data	07-AppMgmt-04

Table 4.4: Threat Table of the CI/CD Building Block

**4.1.3.5.4 Internal Data Store** The Internal Data Store Building Block identifies threats and measures for an application using an internal data store. The technical architecture diagram in Figure 4.6 illustrates the relevant components and interactions: the client, data store, and backup data store, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

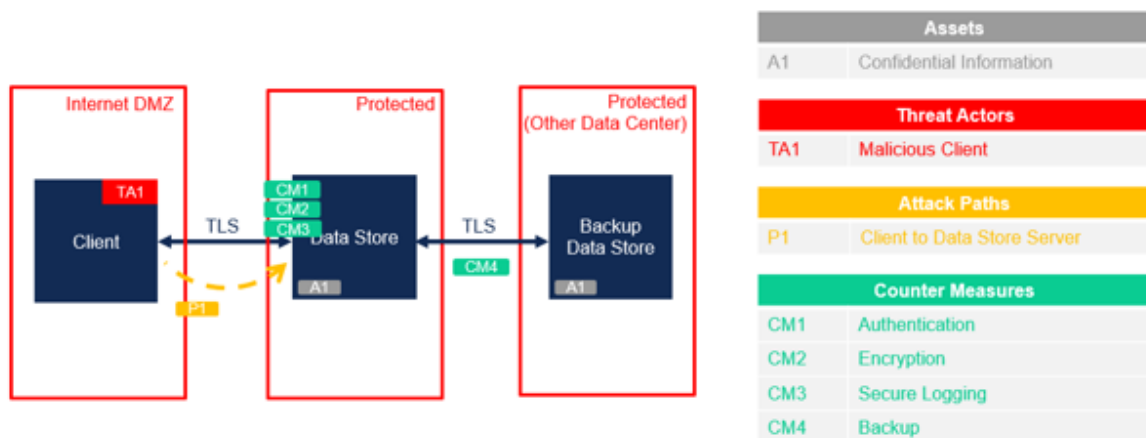


Figure 4.6: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related to authentication, backup, and logging. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.5.

STRIDE	Example of Attack Scenarios	Measures
S	A client impersonating another client to perform action on its behalf	02-Auth-01, 02-Auth-03, 02-Auth-04
T	A client deleting another client's data using stolen credentials	10-Backup-01, 10-Backup-02
R	A client denying modification they performed	02-Auth-01, 04-Log-01, 04-Log-02
I	A client accessing data they are not authorized to see using for example Insecure Direct Object Reference	02-Auth-01, 02-Auth-02, 02-Auth-03, 06-Vuln-08
D	A client consuming the entire bandwidth by requesting or storing large files	No measures in the IS_Schutzbedarfsermittlung
E	A client gaining unauthorized access to higher privileges on the data store server through a Remote Code Execution vulnerability	06-Vuln-01, 06-Vuln-02

Table 4.5: Threat Table of the Internal Data Store Building Block

**4.1.3.5.5 Internet Facing Web Application – On-Premises** The On-Premises Building Block identifies threats and measures for application that are Internet Facing and deployed on-premises. The technical architecture diagram in Figure 4.7 illustrates the relevant components and interactions: the user, web application, and database, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

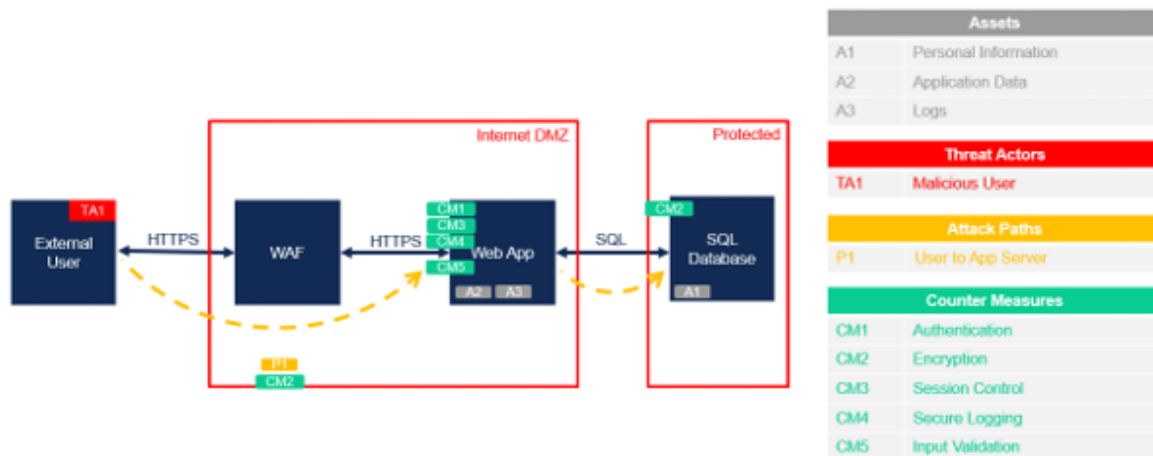


Figure 4.7: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related to input validation, logging, authorization and session control. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.6.

STRIDE	Example of Attack Scenarios	Measures
S	An external user impersonating another user. Example: stolen credentials, weak authentication	02-Auth-01, 02-Auth-03, 02-Auth-04
T	An external user might access another user's data using XSS	08-Validation-01, 08-Validation-02
R	An external user might deny actions they performed	02-Auth-01, 04-Log-01, 04-Log-02
I	A misconfigured access controls could expose sensitive files	02-Author-01, 02-Author-02, 02-Author-03
D	An external user could overwhelm the web server with resource intensive requests	05-Perimeter-01, 05-Perimeter-02
E	An external user could gain unauthorized access to higher privileges through a Remote Code Execution	06-Vuln-01, 06-Vuln-02

Table 4.6: Threat Table of the On-Premises Building Block

**4.1.3.5.6 Cloud** The Cloud Building Block identifies threats and measures for application that are Internet Facing and deployed on the Cloud. The technical architecture diagram in Figure 4.8

illustrates the relevant components and interactions: the user, application pod, data storages, many cloud related components, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

Figure 4.8: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related to encryption, logging and authorization. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.7.



STRIDE	Example of Attack Scenarios	Measures
S	Impersonate the web server to intercept user data. Example: MitM	01-Encrypt-04
T	Unauthorized modifications of logs	04-Log-04
R	The compromised server might deny actions such as accessing the database or files	04-Log-05, 04-Log-06
I	Misconfigured storage accounts or access policies could lead to unauthorized access to data stored in the cloud	02-Author-01, 02-Author-02, 02-Author-03
D	An attacker could target the resource quotas of the AKS cluster to exhaust CPU, memory, or network resources, leading to service degradation or unavailability	05-Perimeter-02
E	Gaining unauthorized access to higher privileges	06-Vuln-01, 06-Vuln-02

Table 4.7: Threat Table of Cloud Building Block

**4.1.3.5.7 Regulatory** The Regulatory Building Block identifies threats and measures related to data regulation, particularly in the case of a server sending data for internal usage in another region. The technical architecture diagram in Figure 4.9 illustrates the relevant components and interactions: the anonymization server that is sharing the data, the database, and the internal usage servers, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

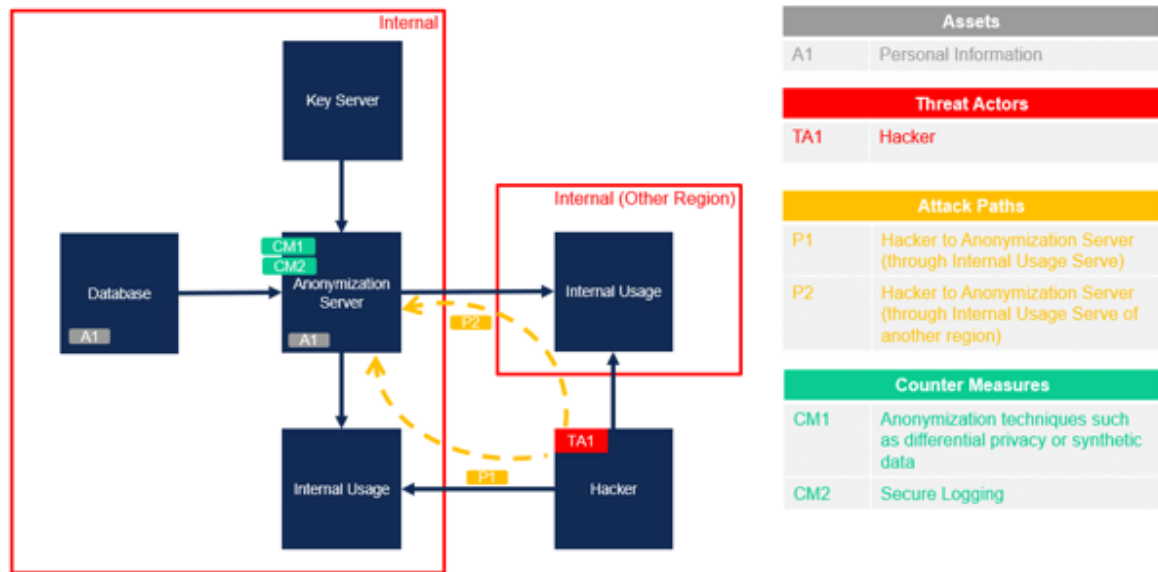


Figure 4.9: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related encryption, logging and vulnerability assessment. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.8.

STRIDE	Example of Attack Scenarios	Measures
S	An attacker might spoof one of the “usage” servers to gain access to more information than allowed	01-Encrypt-04, 01-Encrypt-05
T	N/A	N/A
R	N/A	N/A
I	If the anonymization techniques are not used correctly, the anonymization server might leak personal data which could result in non-compliance with data protection regulations	04-Log-13, 06-Vuln-05, No measures in the IS_Schutzbedarfsermittlung about anonymization but this isn't directly the responsibility of developers
D	Disruption of the anonymization service could prevent the timely processing and transfer of data, affecting compliance with data handling timelines	05-Perimeter-02
E	If an attacker gains elevated privileges within the anonymization service, they could access all the data	06-Vuln-01, 06-Vuln-05

Table 4.8: Threat Table for the Regulatory Building Block

**4.1.3.5.8 Resilience** The Resilience Building Block identifies threats and measures related to the availability and recovery of an application. The technical architecture diagram in Figure 4.10 illustrates the relevant components and interactions: the user, the primary site, failover site, the replicated and backup data, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

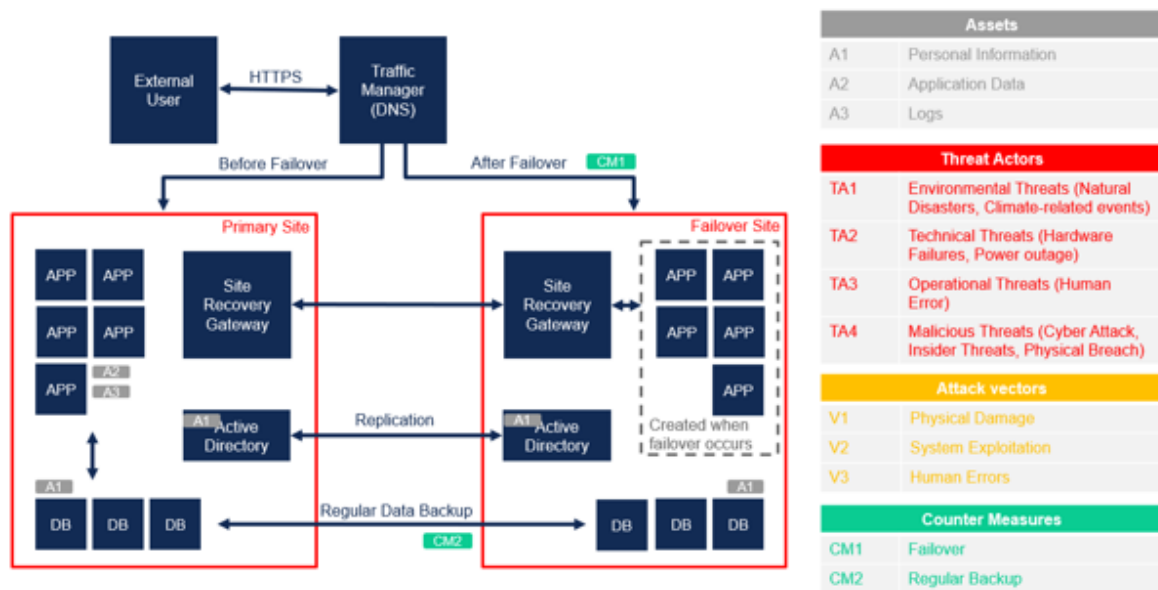


Figure 4.10: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related to backup such as backup verification and storage location. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.9.

STRIDE	Example of Attack Scenarios	Measures
S	N/A	N/A
T	An application's critical data is lost due to an incident that results in the shutdown of the current deployment site	10-Backup-01, 10-Backup-02, 10-Backup-03
R	N/A	N/A
I	N/A	N/A
D	An application becoming unavailable due to an incident that results in the shutdown of the current deployment site	Failover is handled by the DevOps Team
E	N/A	N/A

Table 4.9: Threat Table of the Resilience Building Block

**4.1.3.5.9 Secret Management** The Secret Management Building Block identifies threats and measures for the secrets management process. The technical architecture diagram in Figure 4.11

illustrates the relevant components and interactions: the Vault Server, the user responsible for secrets, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

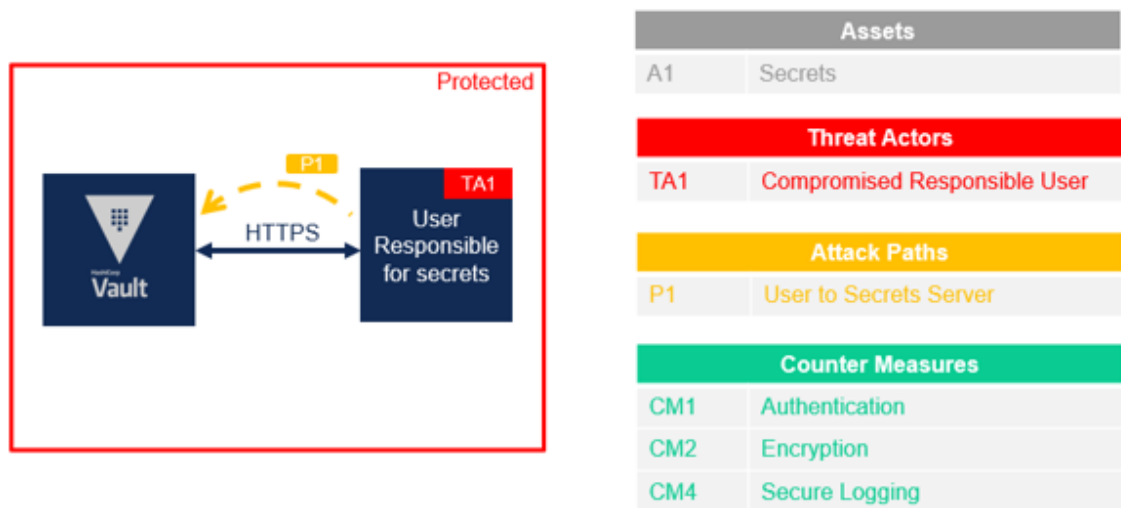


Figure 4.11: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. However, those scenarios are all mitigated with measures handled by the DevOps team. The threats are shown in Table 4.10.

STRIDE	Example of Attack Scenarios	Measures
S	An application pretending to be another application to access its secrets	Security Handled by the DevOps Team
T	Unauthorized alteration of the access policy by a developer	Security Handled by the DevOps Team
R	A developer denies having modified and/or accessed to secrets	Security Handled by the DevOps Team
I	An application has access to secrets from another application because of a misconfiguration in the access policy	Security Handled by the DevOps Team
D	Overwhelming the secrets storage server causing all dependent applications to be out of service	Security Handled by the DevOps Team
E	N/A	N/A

Table 4.10: Threat Table of the Secret Management Building Block

**4.1.3.5.10 Secure Communication** The Secure Communication Building Block identifies threats and measures for encrypted communications. The technical architecture diagram in Figure 4.12 illustrates the relevant components and interactions: the sender, the recipient, the secret key, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

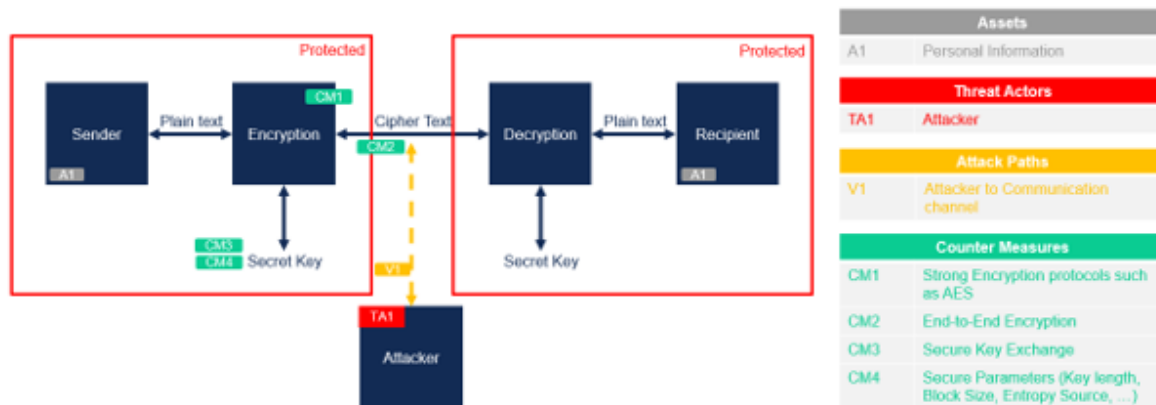


Figure 4.12: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related to cryptography such as using secure protocols and verify certificates with trusted authorities. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.11.

STRIDE	Example of Attack Scenarios	Measures
S	The attacker could impersonate the sender to provide the recipient with false information or malicious content	01-Encrypt-05
T	The attacker could alter messages in transit if they are not properly authenticated, potentially changing the content before it reaches the recipient	01-Encrypt-01, 01-Encrypt-04
R	N/A	N/A
I	The attacker could potentially gain access to the plaintext message if they can break the encryption or obtain the symmetric key	01-Encrypt-01, 01-Encrypt-04
D	N/A	N/A
E	N/A	N/A

Table 4.11: Threat Table of the Secure Communication Building Block

**4.1.3.5.11 Software Supply Chain** The Software Supply Chain Building Block identifies threats and measures linked to libraries in the dependency chain of the application. The technical architecture diagram in Figure 4.13 illustrates the relevant components and interactions: the CI/CD servers, the libraries in the dependency chain, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.

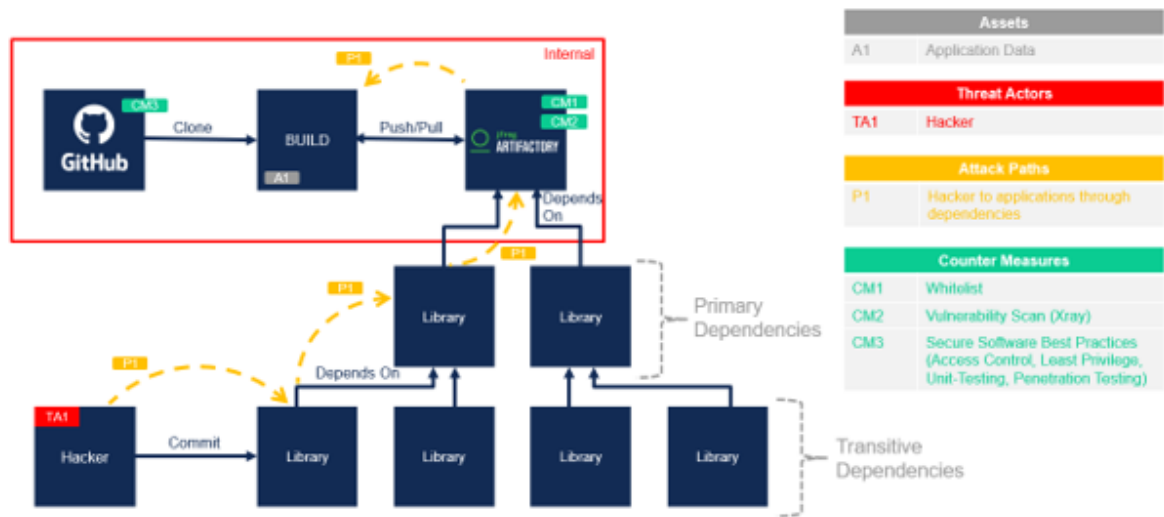


Figure 4.13: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related to hardening, applications management, and developments practices. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.12.

<b>STRIDE</b>	<b>Example of Attack Scenarios</b>	<b>Measures</b>
S	N/A	N/A
T	The vulnerable dependency, once integrated into the main application, could change the application's data processing pipeline	06-Vuln-08, 07-AppMgmt-04, 09-Dev-13
R	N/A	N/A
I	The usage of a library that is not hardened against side channels attacks could make the application in turn vulnerable to side channels attacks	06-Vuln-01, 06-Vuln-08, 09-Dev-13
D	The vulnerable code within the dependency could be used to disrupt services by either directly affecting their functionality or by making them unbearably slow	06-Vuln-01, 06-Vuln-08, 09-Dev-13
E	The vulnerable dependency, once integrated into the main application, could be used to gain elevated privileges	06-Vuln-01, 06-Vuln-08, 09-Dev-13

Table 4.12: Threat Table of the Software Supply Chain Building Block

**4.1.3.5.12 Third Party Service** The Third-Party Service Building Block identifies threats and measures for application that work with third-party services. The technical architecture diagram in Figure 4.14 illustrates the relevant components and interactions: the user, application, and third-party service, as well as the flow of data between these components. The diagram also identifies the critical assets, the potential threat actors, the attack paths, and counter measures.



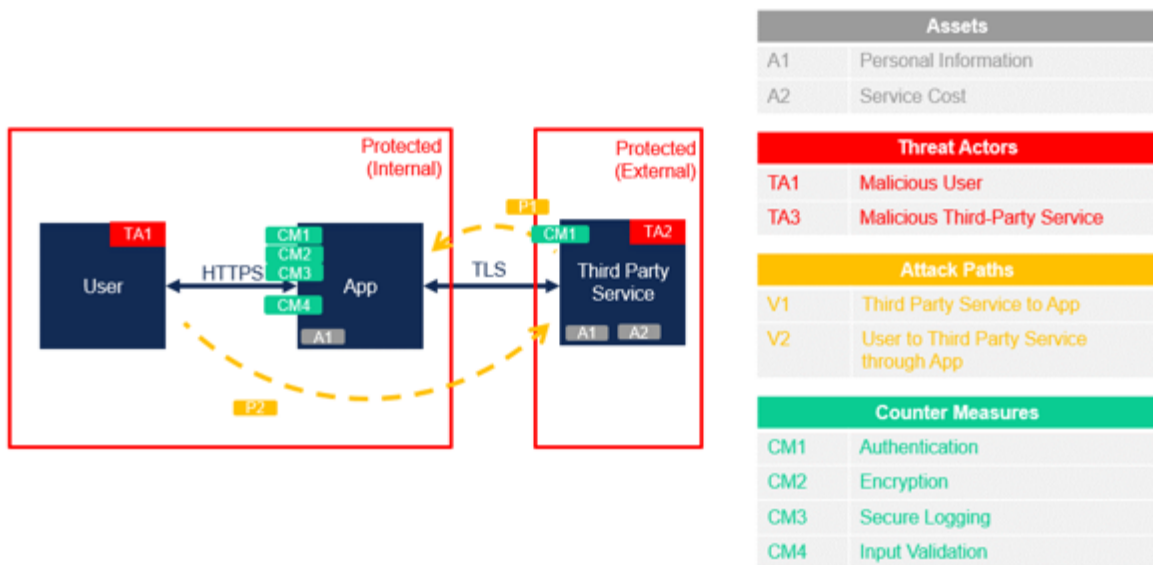


Figure 4.14: This is the caption for the full-width image

Using the technical architecture diagram and the STRIDE methodology, several example attacks scenarios are identified. Those scenarios are mitigated with measures related encryption and logging. The threats with the IDs of the relevant measures from the IS\_Risikobewertung are shown in Table 4.13.

STRIDE	Example of Attack Scenarios	Measures
S	An attacker could spoof the third-party service to send malicious data to the web application or to intercept data sent from the web application	01-Encrypt-05
T	An attacker could tamper with the data being exchanged between the	01-Encrypt-04
R	The third-party service might deny having received or sent certain data/instruction	04-Log-02, 04-Log-05
I	Sensitive data from the web application could be exposed through an insecure third-party	No measures against Third-Party Failures but it is not the responsibility of developers
D	If the third-party service is targeted and becomes unavailable, it could affect the functionality of the web application that relies on it	No measures against Third-Party Failures but it is not the responsibility of developers
E	If the third-party service has vulnerabilities, an attacker could leverage these to gain elevated access to the web application	06-Vuln-01

Table 4.13: Threat Table of the Third-Party Service Building Block

#### 4.1.3.6 Artifacts and Verification

This section provides a checklist for developers to confirm that all necessary steps in the threat modeling methodology are completed. Each item on the checklist must be checked off by each responsible application team to confirm completion.

- ☐ Understand the methodology: Read and understand the threat modeling methodology as described in Section 3.
- ☐ Review IS\_Schutzbedarfsermittlung questionnaire: Identify the critical assets and understand regulatory requirements from the IS\_Schutzbedarfsermittlung questionnaire answers.
- ☐ Read all architecture building blocks: Understand the purpose and application of the architecture building blocks presented in Section 4.1.3.5.
- ☐ Select relevant building blocks: Choose the architecture building blocks that correspond to the specific features and scenarios of the application.

- ☐ Assess exploitability of threats: Consider whether any threat of the building blocks is exploitable due to a vulnerability within the application.

## 4.2 Reference Security Architectures

### 4.2.1 Theoretical Background

Reference Security Application Architectures are standardized templates that guide the development of software systems. These templates are blueprints, detailing necessary components and their interactions for constructing secure applications. By systematically using the same architectures across projects, developers use a uniform approach to development, thereby minimizing security vulnerabilities due to inconsistent design practices.

The primary objectives of Reference Security Application Architectures are:

1. **Robust Security Foundation:** Establish a strong security foundation from the beginning, integrating security considerations into the design phase rather than as an afterthought.
2. **Simplified Development Process:** Provide a clear and structured framework that simplifies the development process for new projects or features, allowing developers to focus more on functionality while ensuring security.
3. **Compliance with Standards:** Ensure that software development processes comply with internal and external security standards and regulations.
4. **Facilitate Communication:** Enhance collaboration among development, security, DevOps, Cloud, and Infrastructure teams by providing a common reference point for security practices and expectations.

### 4.2.2 Requirements

The requirements for this part are straightforward. Insurance CH is responsible for creating, approving, and documenting a set of reference application architectures. Additionally, they must ensure the continuous maintenance and enhancement of these reference architectures, incorporating new insights and developments within the organization.

### **4.2.3 Solution**

Our solution consists of two key components. First, it defines the responsibilities clearly to ensure accountability. Second, it provides the initial set of Reference Security Application Architectures to establish a foundation and guide future developments.

#### **4.2.3.1 Responsibilities**

Clearly delineating roles and responsibilities of the various teams is important to ensure the Reference Security Application Architectures are effectively developed, maintained, and utilized. This section outlines the key responsibilities for different stakeholders involved.

Architects and Senior Developers are responsible for designing, documenting, and continuously improving the reference security application architectures. They collaborate to ensure these architectures incorporate best security practices and provide clear and accessible documentation for all relevant stakeholders. Additionally, they gather feedback from development teams and conduct regular reviews to update the architectures based on new insights and technological advancements.

Application Development Teams adopt the reference application architectures in their projects to ensure consistency and security. They follow the documented architectures, provide feedback based on practical implementation experiences, and suggest improvements.

#### **4.2.3.2 Reference Security Application Architectures**

**4.2.3.2.1 Web Application Reference Security Architecture** The Web Application Reference Security Architecture focuses on establishing a strong foundation for applications that are accessed via web browsers. This architecture delineates a multi-layered structure for Internet-facing applications, as depicted in Figure 4.15. It comprises three main application layers—web-tier, business-tier, and data-tier—along with a Web Application Firewall (WAF) and the internet layer. This layered approach ensures robust defense across the application. The internet layer ensures secure data transmission via protocols like HTTPS. The WAF layer provides an additional security checkpoint, filtering traffic to prevent some web-based attacks. The web-tier serves the user interface, managing sessions and user authentication. The business-tier processes requests according to business requirements. The data-tier handles secure data storage and retrieval.

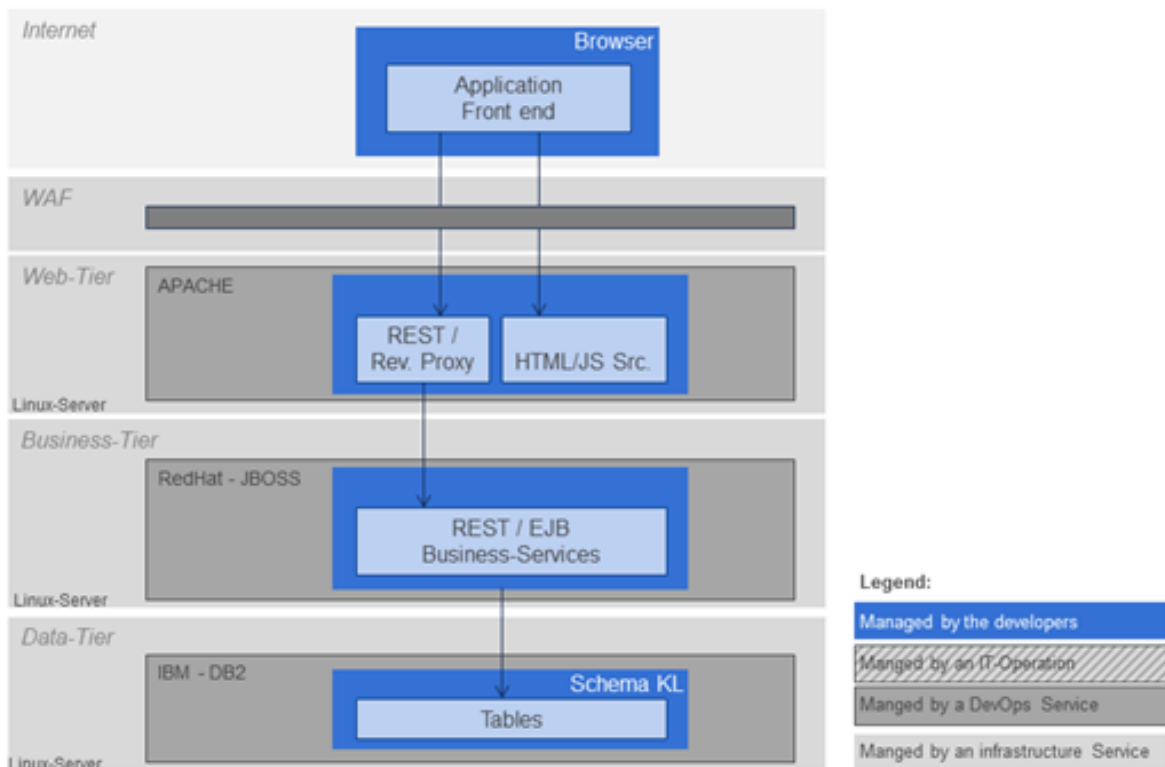


Figure 4.15: Web Application Reference Security Architecture (Overview)

Additionally, within the frontend, business, and data layers, various components interact to enhance the application's functionality and security, as depicted in Figure 4.16:

- Angular Controller: Manages the user interface and user interactions.
- Service: Acts as an intermediary between the Angular Controller and the Business Layer, handling HTTP requests and responses securely.
- Transaction Manager: Ensures that operations are completed correctly and consistently, maintaining data integrity.
- Entities: Represent the data model, ensuring that data structures are well-defined and can be protected effectively.
- Business Rules: Enforces the business logic and rules, ensuring compliance with organizational policies and regulations.
- Validation: Ensures that data meets defined criteria before processing, preventing invalid data from entering the system.

- Data Access Object (DAO): Provides an abstract interface to the database, ensuring secure data access and manipulation.
- Mapper: Transforms data between the database format and the application format, ensuring data integrity.
- Database: Stores data securely.

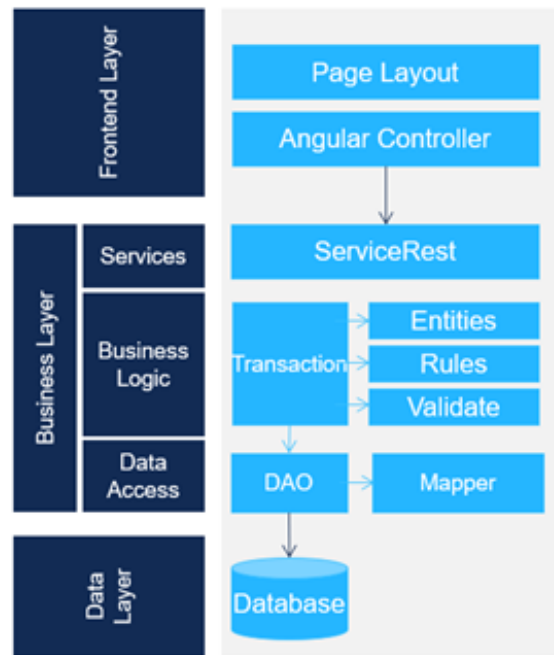


Figure 4.16: Web Application Reference Security Architecture (Layers Details)

**4.2.3.2.2 Authentication Reference Security Architecture** The Authentication Reference Security Architecture provides an overview of the authentication mechanisms within applications. The architecture diagram in Figure 4.17 illustrates the relevant components and interactions: the user, the application, and the authentication server, as well as the flow of data between these components. The arrows indicating the data flow are numbered to reflect the sequence of actions. In this architecture, a user attempting to access an application is redirected to an authentication server for credential verification. Upon successful authentication, the server redirects the user back to the application. This seamless redirection and verification process ensures that only authenticated users gain access to the application without each application needing its own authentication system.

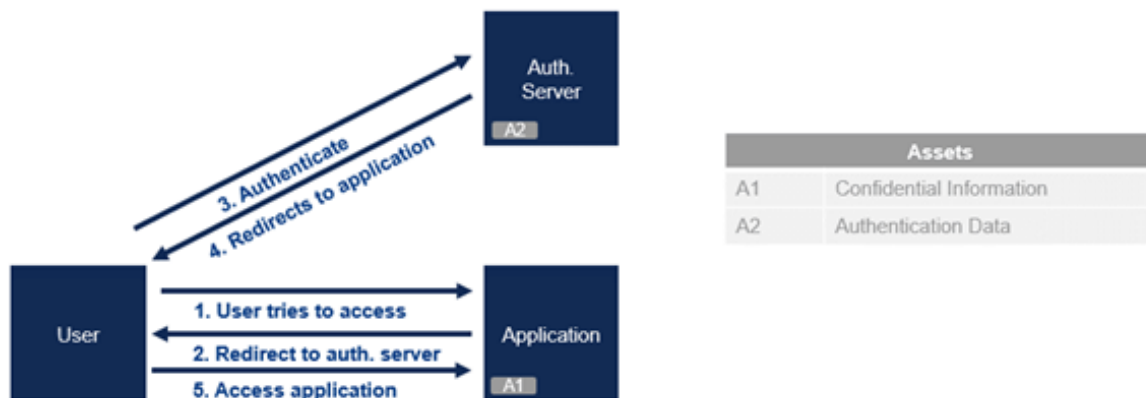


Figure 4.17: Authentication Reference Security Architecture

**4.2.3.2.3 Certificate Management Reference Security Architecture** The Certificate Management Reference Security Architecture outlines the use of digital certificates for secure communications between applications and users. The architecture diagram in Figure 4.18 depicts the relevant components and interactions, including the Certificate Authority (CA), the developer, the application, the end-user, along with the flow of data. The arrows indicating the data flow are numbered to reflect the sequence of actions. In this architecture, the process begins with a developer creating a Certificate Signing Request (CSR) for its application. Then, the developer sends the certificate to the Certificate Authority (CA). Once the CA receives the CSR, it verifies the legitimacy of the request and issues a digital certificate to the developer. The developer then installs this certificate on the application server. When a user initiates an HTTPS connection to the application, the server sends the digital certificate to the user's browser. The browser verifies the certificate's authenticity and, upon successful verification, establishes a secure connection, allowing the user to access the application securely.

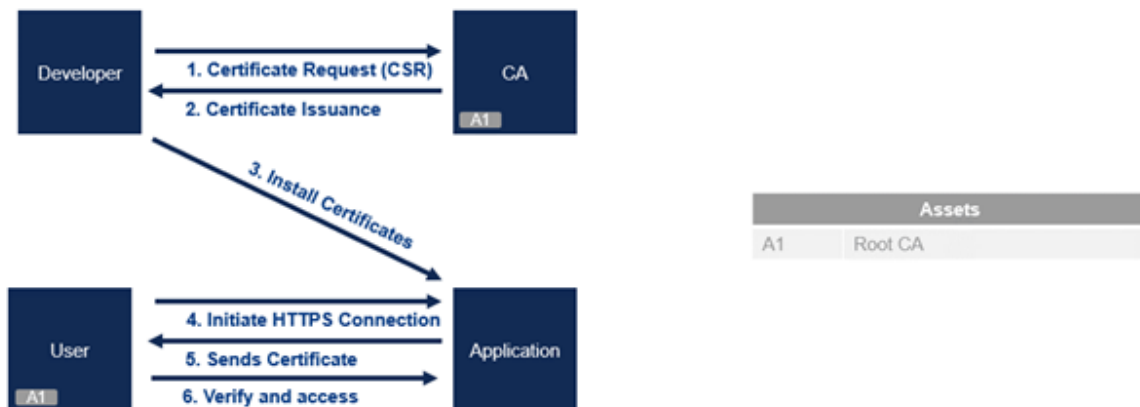


Figure 4.18: Certificate Management Reference Security Architecture

**4.2.3.2.4 Data Encryption and Key Management Reference Security Architecture** The Data Encryption and Key Management Reference Security Architecture provides an outline for managing cryptographic keys to protect sensitive data. The architecture diagram in Figure 4.19 illustrates the relevant components and interactions: the application, the developer, the secret vault, and the data store, as well as the flow of data between these components. The arrows indicating the data flow are numbered to reflect the sequence of actions. In this architecture, the process begins with a developer generating and uploading a cryptographic key to HashiCorp Vault using a secure HTTPS connection. The application then securely retrieves the key from HashiCorp Vault to encrypt and decrypt data in the data store. This ensures that key management is centralized and secure, protecting sensitive data from unauthorized access.

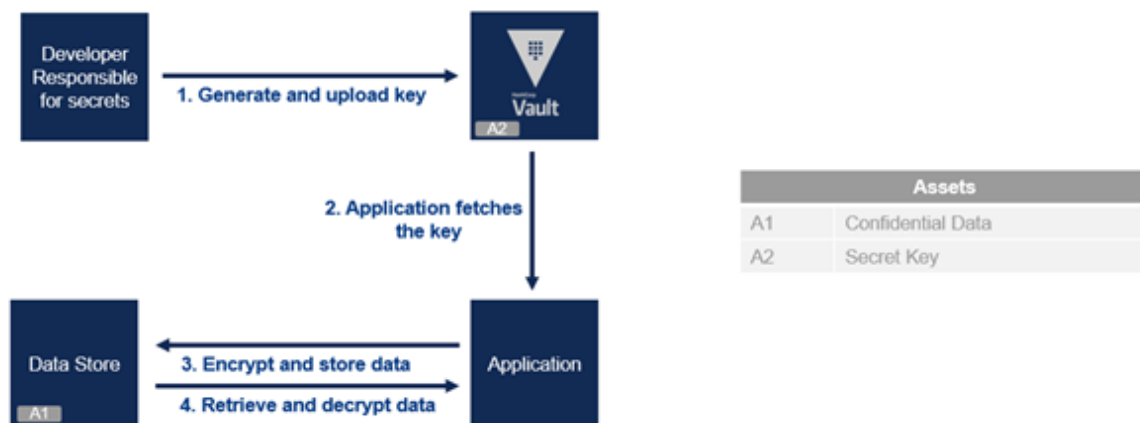


Figure 4.19: Data Encryption and Key Management Reference Security Architecture



**4.2.3.2.5 DevOps Reference Security Architecture** The DevOps Reference Security Architecture provides an outline of the different elements and security practices in the DevOps infrastructure. The architecture diagram in Figure 4.20 illustrates the relevant components and interactions: the developer, the GitHub Server, the Jenkins server, the JFrog Artifactory server, and the production and staging servers. In this architecture, developers securely push and pull code from GitHub. Jenkins then pulls the code from GitHub to build and deploy to the staging or production server. Jenkins operates in an air-gapped environment and can only fetch third-party dependencies from JFrog Artifactory, ensuring that only approved dependencies are used. Finally, developers can retrieve logs from the production and staging servers, maintaining a secure and efficient development pipeline.

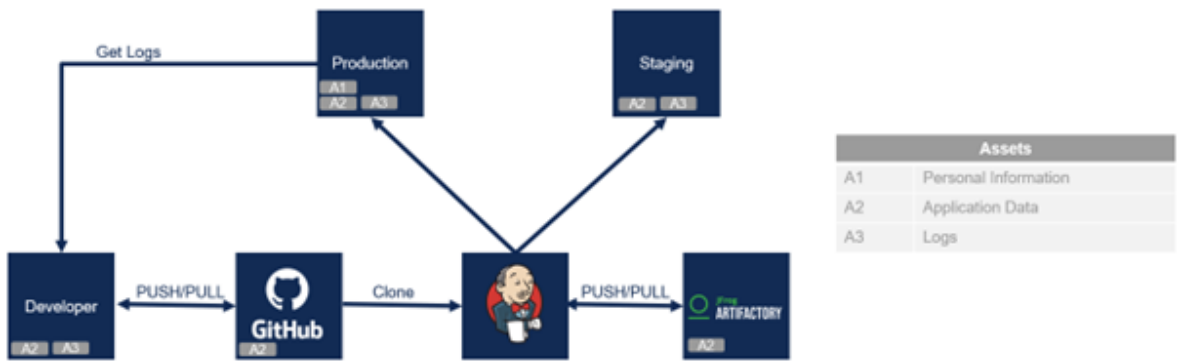


Figure 4.20: DevOps Reference Security Architecture

## Chapter 5

# Conclusion

This master's project set out to enhance the secure software development practices at Insurance CH by designing and integrating custom security solutions as part of a broader cyber transformation program. Through multiple subprojects, this project addressed key deficiencies and established robust frameworks for secure software practices, ultimately aligning the organization's development processes with the global Information Security Policy.

### 5.1 Summary of Results

The project successfully met its objectives by delivering three key subprojects, each addressing a specific aspect of secure software development:

1. **Third-Party Libraries Management:** This subproject established a secure and compliant framework for managing third-party libraries within the organization's software development processes. By integrating JFrog Xray with Artifactory and defining clear onboarding and maintenance processes, the project mitigated risks associated with using third-party components. It ensured that all dependencies met the organization's security standards, reducing potential vulnerabilities.
2. **Secret Management:** The project finalized and operationalized the use of HashiCorp Vault for managing secrets. By implementing a structured namespace, role-based access control (RBAC) system, and integrating with Azure Active Directory, the solution provided a robust framework for securing and managing secrets. This not only closed significant compliance gaps but also improved the overall security of the organization's applications and infrastructure.
3. **Threat Modeling and Reference Security Architectures:** This initiative focused on providing developers with tools and methodologies to identify, assess, and mitigate potential threats

early in the software development lifecycle. By developing architecture building blocks and reference security architectures, the project enables developers to proactively address security concerns, fostering a culture of security-first development.

Collectively, these subprojects have significantly enhanced the maturity and security of software development practices at Insurance CH. They have provided a scalable and sustainable foundation for secure development that aligns with the strategic objectives of the organization and ensures compliance with the Information Security Policy.

## 5.2 Key Insights Across the Project

This master's project revealed several key insights essential for strengthening secure software practices at Insurance CH.

1. **Balancing Security and Usability:** The project highlighted the importance of balancing strong security measures with operational ease. The initial Third-Party Library Management solution was rejected for being too complex, leading to a more streamlined approach that still met security requirements without overburdening operations. This emphasizes the need for solutions that are both secure and practical.
2. **Collaboration and Knowledge Sharing:** Success depended on effective collaboration and communication among teams, such as development, DevOps, and Enterprise Architecture Teams. Implementing HashiCorp Vault for secret management required coordinated efforts for smooth adoption. Additionally, thorough documentation and sharing of best practices helps ensure adoption across all teams.
3. **Leveraging Existing Tools:** Utilizing existing infrastructure and tools, such as JFrog Artifactory and HashiCorp Vault, proved effective in enhancing security without introducing unnecessary complexity. This approach minimized costs and disruptions, while ensuring a smoother adoption process by leveraging familiar technologies.
4. **Fostering a Security-First Culture:** The project underscored the value of embedding security into every phase of development and encouraging proactive involvement from all stakeholders. This approach helped cultivate a culture where security is prioritized and ingrained in the organization's daily practices.

In summary, this project highlighted the importance of a balanced, collaborative, and adaptable approach to cybersecurity.

## 5.3 Next Steps

While the project has achieved progress towards the overarching transformation, several areas require continued focus and further development to maintain and build upon the gains achieved:

1. **Continuous Improvement and Monitoring:** The solutions implemented during this project, particularly those involving third-party libraries management and secrets management, require ongoing monitoring to ensure they work smoothly.
2. **Expansion of Threat Modeling and Reference Architectures:** Although the initial set of threat modeling building blocks and of reference architectures have been successfully deployed, there is a need to expand this to cover a broader range of scenarios and technologies. Future efforts should involve creating more detailed and comprehensive threat models and reference architectures, especially as the organization adopts new technologies.
3. **Addressing Remaining Security Requirements:** Continue developing solutions to meet other requirements from the Information Security Policy. This includes, for example, designing a solution for Dynamic Application Security Testing (DAST) to identify vulnerabilities in business applications and establishing a robust code signing solution to ensure the integrity and authenticity of code before deployment.

By following these next steps, Insurance CH can build upon the foundational work completed during this project, ensuring a secure, compliant, and efficient software development environment that supports the organization's broader business objectives.

# Bibliography

- [1] *Estimated cost of cybercrime worldwide 2018-2029 (in trillion U.S. dollars) [Graph]*. <https://www.statista.com/forecasts/1280009/cost-cybercrime-worldwide>. June 2024.
- [2] World Bank. *World Development Indicators: GDP (current US\$)*. Accessed: 2024-07-18. 2024. URL: <https://data.worldbank.org/indicator/NY.GDP.MKTP.CD>.
- [3] *2024 Climate and Catastrophe Insight*. Accessed pages 11 and 16. Aon, 2024. URL: <https://www.aon.com>.
- [4] S-RM Intelligence and Risk Consulting. *Cyber Insights Report*. 2022. URL: <https://www.s-rminform.com/cyber-security-insights-report-2022>.
- [5] Kevin Eiden, James Kaplan, Bartlomiej Kazimierski, Charlie Lewis, and Kevin Telford. "Organizational cyber maturity: A survey of industries". In: *McKinsey Insights* (Aug. 2021). URL: <https://www.mckinsey.com/capabilities/risk-and-resilience/our-insights/organizational-cyber-maturity-a-survey-of-industries>.
- [6] S-RM Intelligence and Risk Consulting. *Cyber Insights Report 2023*. 2023. URL: <https://www.s-rminform.com/news/cyber-insights-report-2023-press-release>.
- [7] ISC2. *ISC2 2023 Cybersecurity Workforce Study*. Oct. 2023. URL: <https://www.isc2.org/research>.
- [8] ISACA. *An Executive View of Key Cybersecurity Trends and Challenges in 2023*. 2023. URL: <https://www.isaca.org/resources/news-and-trends/industry-news/2023/an-executive-view-of-key-cybersecurity-trends-and-challenges-in-2023>.
- [9] Verizon. *2024 Data Breach Investigations Report: Summary of Findings*. Tech. rep. Verizon, 2024. URL: <https://www.verizon.com/business/resources/reports/dbir/2024/summary-of-findings/>.
- [10] SolarWinds Corporation. *Form 8-K Current Report*. Accessed: 2024-08-28. 2020. URL: <https://www.sec.gov/ix?doc=/Archives/edgar/data/1739942/000162828020017451/swi-20201214.htm>.
- [11] Wiz and EY. *Enterprises halfway through patching Log4Shell*. 2021. URL: <https://www.wiz.io/blog/10-days-later-enterprises-halfway-through-patching-log4shell>.
- [12] *Jenkins*. Jenkins Project. 2024. URL: <https://www.jenkins.io>.

- [13] *JFrog Artifactory*. JFrog. 2024. URL: <https://jfrog.com/artifactory/>.
- [14] *npm*. npm, Inc. 2024. URL: <https://www.npmjs.com>.
- [15] Jenkins Project. *Managing Nodes*. 2024. URL: <https://www.jenkins.io/doc/book/managing/nodes/>.
- [16] *JFrog Xray*. JFrog. 2024. URL: <https://jfrog.com/xray/>.
- [17] *HashiCorp Vault*. Accessed: 2024-08-28. HashiCorp. 2024. URL: <https://www.vaultproject.io>.
- [18] OWASP. *Hardcoded Secrets*. Accessed: 2024-08-28. 2024. URL: [https://owasp.org/www-community/vulnerabilities/Hard\\_coded\\_secrets](https://owasp.org/www-community/vulnerabilities/Hard_coded_secrets).
- [19] Microsoft. *Azure Active Directory*. 2024. URL: <https://docs.microsoft.com/en-us/azure/active-directory/>.
- [20] OpenID Foundation. *OpenID Connect Core 1.0*. 2024. URL: [https://openid.net/specs/openid-connect-core-1\\_0.html](https://openid.net/specs/openid-connect-core-1_0.html).
- [21] Kubernetes Project. *Kubernetes*. 2024. URL: <https://kubernetes.io>.
- [22] HashiCorp. *Vault CSI Driver*. 2024. URL: <https://www.vaultproject.io/docs/platform/k8s/csi>.
- [23] John Steven. “Threat Modeling - Perhaps It’s Time”. In: *IEEE Security Privacy* 8.3 (2010), pp. 83–86. DOI: 10.1109/MSP.2010.110.
- [24] OWASP. *Threat Modeling Process*. Accessed: 2024-08-28. 2024. URL: [https://owasp.org/www-community/Threat\\_Modeling\\_Process#stride](https://owasp.org/www-community/Threat_Modeling_Process#stride).
- [25] Sonatype. *Maven Central Repository*. 2024. URL: <https://mvnrepository.com/repos/central>.
- [26] P. Torr. “Demystifying the threat modeling process”. In: *IEEE Security Privacy* 3.5 (2005), pp. 66–70. DOI: 10.1109/MSP.2005.119.
- [27] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.