William Truc Truong

Enhancing Cyber Security in Ecommerce Development

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master’s Thesis

8 May 2025

Abstract

Author:	William Truc Truong
Title:	Enhance Cybersecurity in Ecommerce Development
Number of Pages:	34 pages + 0 appendices
Date:	1 May 2025
Degree:	Master of Engineering
Degree Programme:	Information Technology
Professional Major:	Networking and Services
Supervisors:	Antti Piironen, Principal Lecturer

The study explores organised process for enhancing cyber security in ecommerce development by integrating a set of modern cyber security practices and tools. Since the pandemic, ecommerce has gained more popularity among consumers and the cyber threats also thrived at faster pace than ever before bringing more challenges to securing customer data and ensuring system integrity [1]. The study focuses on methodologies for applying automated processes such as dependency management, integrating static code analysis into the software development of ecommerce platform, as well as applying industry standard security principles. By focusing on practical applications, the study provides a framework to define, analyse and reduce the vulnerabilities throughout the development and operation of the ecommerce. The findings of the study will help to provide the actionable details and scalable solutions for developers and enterprises, contributing to the ongoing effect to build a safe, resilient and trustworthy ecommerce platform.

Keywords: ecommerce security, secure software development

The originality of this thesis has been checked using Turnitin Originality Check service.
ChatGPT was used to assist with grammar correction and language refinement during the writing process. All content was reviewed and edited by the author to ensure accuracy and relevance.

Contents

List of Abbreviations

API	Application Programming Interface
CI/CD	Continuous Integration/Continuous Deployment
CIA	Confidential, Integrity, and Availability
DAST	Dynamic Application Security Testing
GDPR	General Data Protection Regulation
Jira	Jira Software (Project Management Tool)
KPI	Key Performance Indicator
MFA	Multi-Factor Authentication
NPM	Node Package Manager
OWASP	Open Web Application Security Project
RBAC	Role-Based Access Control
SAT	Static Application Security Testing
SDLC	Software Development Life Cycle
Semgrep	Static Code Analysis Tool
Snyk	Security Scanning tool for Dependencies
SSO	Single Sign-On

TLS	Transport Layer Security
Zero Trust	Zero Trust Security Model
Dependabot	Automated dependency Management Tool
Auth0	Authentication and Authorization Platform
Azure AD	Azure Active Directory
SonarQube	Static Code Analysis Tool

1 INTRODUCTION

E-commerce has become one of the most essential parts of modern digital economies, offering availability around the clock as well as convenience for consumers. It has also significantly transformed how the companies operate and created new possibilities for customer relationships and spaces to explore new ideas.

While the e-commerce revenue worldwide has continued to surge in the last few years [1], cyber security risks have also escalated at an alarming rate. Security risks such as data breaches, vulnerabilities in third-party dependencies or supply chain attacks have become increasingly visible and critical for businesses. If companies do not address to these security risk seriously, they risk losing revenue, reputation and customer trust. This study is based on company X, a major home improvement retailer in Finland, ensuring practical relevance while maintaining confidentiality.

Hackers often target e-commerce platforms due to the vast amount of sensitive customer information they store, including names, addresses, IP addresses, email addresses or financial details. Once hackers identify the system vulnerabilities, they can exploit them to gain unauthorised access to data, leading to financial loss, reputation damage, and legal consequences for companies and customers. To prevent this from happening, e-commerce companies need to utilise stronger and robust security processes.

Even though security tools and technologies are continuously improving, many companies still have security issues as they often do not update their dependencies, use insecure coding practices, or ineffective manual security checks. Manual security checks are time-consuming and often fail to detect vulnerabilities early in the development cycle. Therefore, companies need an automated process in security to work smoothly with their development processes. This thesis will focus on how to improve cyber security in ecommerce

using automated dependency management, static application security testing with code security scanning tools as well as stronger security principles.

The goal of this study is to examine and implement a structured security process in e-commerce platform development. Specifically, the study includes:

- Checking how automated dependency management can help reduce security vulnerabilities
- Finding out how static application security testing (SAST) with code security scanning tools can help detect security vulnerabilities before they become visible
- Exploring how security principles, such as the least privilege model and secure coding practices can enhance e-commerce security
- Developing a framework that can be adopted by other teams to enhance e-commerce security

The thesis focuses on improving cyber security for e-commerce platforms, especially on how companies handle their obsolete dependencies, secure their code base, and apply security principles. This study is based on a case study of company X, a major home improvement retailer in Finland, to ensure practical relevance. The study does not attempt to cover all topics in cybersecurity. It does not focus on social engineering attacks, customer authentication or payment fraud. Instead, the study focuses on software security measures that e-commerce businesses can implement to create a more secure environment.

2 THEORETICAL FOUNDATIONS AND ANALYSIS OF EXISTING CYBER SECURITY IN ECOMMERCE

Digital transformation has propelled e-commerce into becoming one of the most important components of the global economy. E-commerce provides users with instant access to goods and services, facilitates cashless transactions, and enables business to offer a seamless shopping experience [2]. However, as e-

commerce solutions continue to evolve, cybersecurity risks have increased in complexity and scale [1].

Cybersecurity threats in e-commerce are not limited to basic attacks but also include complex challenges such as data breaches, vulnerabilities in third party dependencies, and supply chain attacks [1,4]. Many companies use obsolete software and not set up strong protection, which makes it easy for attackers to break in [2]. According to Verizon's 2023 report, e-commerce is one of the most attacked industries because of the valuable data it holds [10].

To solve these problems, businesses need to use many layers of security such as safer software development, better user authentication, and automated tools to find and fix vulnerabilities [5, 6].

E-commerce systems have many weaknesses that hackers can take advantage of. These weaknesses can be in software, network, or human mistakes [1,2].

Data breaches occur when attackers gain unauthorized access to sensitive customer or company information. Attackers often exploit weaknesses in password policies, cloud configurations, or encryption standards [3]. Once hackers enter the system, they can stead personal details, credit card information, and even modify transactions. A 2024 report by Ponemon Institute revealed that average cost of a data breach in e-commerce exceeds $4.5 million [11]. A notable example is 2022 SHEIN data breach, where hackers exploited vulnerabilities in API security and encryption protocols, leading to the exposure of 39 million customers records. The consequences of such breaches go far beyond financial losses, as they can result in legal action, loss of customer trust, and compliance violations [3,4].

Many e-commerce platforms depend on third-party services such as payment processors, analytics tools, and external libraries. While these integrations enhance platform functionality, they also introduce security vulnerabilities. Attackers often target these third-party services because their security measures might not be as robust as those of the main e-commerce systems [2, 4]. A well-

documented example is Magecart attacks, when cybercriminals insert malicious JavaScript code into compromised websites' payment processing pages, allowing them to capture credit card and details in real time. The risk is further increased when business fail to monitor updates and security patches for their third-party dependencies, leaving them exposed to known vulnerabilities [5, 6].

Instead of attacking the business directly, cybercriminals can penetrate to software suppliers, injecting malicious code into legitimate software updates. These infected updates are distributed to multiple e-commerce platforms, giving attackers access to a big number of systems all at once. The SolarWinds attack (2020) is a typical example how of how supply chain attacks can have a massive impact. Attackers compromised the software update mechanism of SolarWinds' IT monitoring software, affecting thousands of customers worldwide, including major corporations and government agencies. In e-commerce, such attacks can lead to payment fraud, data exfiltration, or unauthorised modifications to website code [6, 7].

Instead of targeting technical vulnerabilities, many attackers rely on human manipulation through phishing emails, fake websites, or social engineering tactics. Phishing attacks involve tricking users into revealing login credentials, financial details, or other sensitive information by impersonating a legitimate organisation. Studies indicate that 90% of successful cyberattacks start with phishing, making it one of the biggest security challenges for e-commerce businesses. Attackers often use convincing emails pretending to be from payment processors, logistics providers, or even internal IT teams, prompting employees to or customers to disclose information. Once credentials are obtained, attackers can gain full control of accounts, steal funds, or distribute malware [8]. To mitigate these risks, businesses should implement anti-phishing training programs, email filtering solutions, and multi-factor authentication (MFA) to make it harder for attackers to exploit compromised credentials [2,9].

2.1 Industry practices and tools

To protect e-commerce systems, organizations often implement various security measures. However, three critical areas including static application security testing (SAST), automated dependency management, and strong authentication mechanisms play an essential role in mitigating security vulnerabilities and platform integrity [2, 5].

SAST is a fundamental security practice that allows developers to identify vulnerabilities in source code before deployment. It scans applications at the static code level, meaning vulnerabilities can be detected at early stages of the development process without executing the code. SAST tools analyze code structures, logic, and known vulnerabilities patterns to flag security risks that could be exploited by attackers. SAST plays a very significant role in the development of e-commerce [6]. Early detection of vulnerabilities reduces the cost and effort required for remediation after deployment. In addition, SAST ensures compliance with industry standards such as OWASP and PCI DSS, which mandate security coding practices for platforms that handle sensitive customer information. By integrating SAST into continuous integration and deployment pipeline, security checks can become a standard part of the development process. Several SAST tools are widely used in e-commerce security, including SonarQube, Checkmarx, and Snyk Code [5,8]. These tools scan application source code and flag issues before they reach production. Integration of such tools into software development helps enhance security and reduce the chances of vulnerabilities being exploited.

Most modern e-commerce platforms rely on third-party libraries and open-source components, which introduce security risks if they are not properly managed and maintained. Automated dependency management is a vital part of security practice to ensure third party software components are updated regularly to patch known vulnerabilities [4]. Outdated dependencies generates significant risks. Vulnerabilities in widely used libraries can be exploited by attackers, leading to data breaches and various system compromises. Compatibility issues may also

arise when old libraries are not updated to align with modern security standards. Furthermore, supply chain attacks, in malicious code is inserted into third-party components, have become a significant concern for software security issues.
To address these risks, business often utilise dependency management tools such as GitHub's Dependabot, Renovate, OWASP Dependency-Check and Snyk. These tools continuously monitor project dependencies for security vulnerabilities and recommend updates when risks are identified [7]. By automating the process, companies can mitigate the threat with minimal manual intervention.

Implementing strong authentication mechanisms is important for protecting user account and sensitive data in e-commerce platforms. Strong authentication significantly reduces the risks of unauthorized access and data breaches.
Multi-Factor Authentication (MFA) is a popular practice the requires users to provides multiple forms of verification to access their accounts, significantly reducing the security risks associated with authentication. By combining something that users know such as password with something user has, such as a mobile device, MFA makes authentication much harder for attackers [3, 5].

The Zero Trust Security model strengthens the authentication by operating under the principles that no user or system should be inherently trusted. Instead, continuous verification and strict access controls ensure should be implemented to ensure that even authorized users must authenticate repeatedly to perform critical actions. This model is even more necessary to e-commerce platforms that operate on cloud services, where remote access and multiple third-party integrations increases the risks of security breaches [6]. By applying this security model, companies can help to reduce the threats within their infrastructure, and reduce the impact of potential breaches [7, 9]. Security principles in software development are important to ensure the resilience and integrity of e-commerce platforms. When these principles are ignored, systems tend to be more vulnerable to attacks, unauthorised access, and data breaches. There are several key principles that guide the development of secure software, ensuring

that security is taken care at the very early stages of development rather than just an afterthought [2, 4].

The principle of Least Privilege is a fundamental security measure that restrict user and system access to only what is necessary to perform assigned tasks. By limiting the number of users with administrative or comparable privileges, the risk of insider threats and exploitation of compromised accounts is significantly minimized. Many security breaches occur because attackers gain administrative access through stolen credentials or social engineering attacks. Having principle of Least Privilege properly implemented can help limit the scope of the damage in the event of the account being compromised. Platforms such as AWS provides tools like Identity and Access Management policies, where granular permissions are assigned based on roles, reducing the risks of unauthorized access [8].

Lastly DevSecOps is security principles that integrates security directly to the DevOps pipeline, ensuring continuous assessments of security during the development and deployment. By embedding security automation tools such as Static Application Testing (SAST) and automatic dependency checks into the CI/CD pipeline, organizations can ensure that vulnerabilities are detected and resolved well before they go to production. DevSecOps help fill the gap between development and security teams allowing the security team to focus on on-going concerns rather than managing the pipelines [10].

2.2 Study gap

During the literature review and research process, it is found that many tools mentioned in the papers are not suitable for small ecommerce platform development teams. Some solutions are very expensive, and many of the open-source tools are not easy to use or do not support modern development well. Most of the good tools require commercial license just to try them, which makes it difficult for companies to test and use them freely. Because of this, the study will not focus on comparing paid tools or commercial solutions.

Also, it is noticed that many frameworks or case studies are made for big companies or just general software systems. They are not specific for ecommerce platforms with microservices [4, 7]. In this thesis, I only focus on the real setup of company X, so it is not possible to make wide comparisons with other companies. The purpose is not to build a general solution for all companies, but to improve one specific case. Some tools like GitHub Advanced Security or Snyk are very popular in theory, but in company X they are not used much in daily work. So, this study does not go deep into advanced usage of those tools. In many studies, it looks like development and security teams work together closely [3,6]. But in this case study, there is almost no connection between them. Security work is not really part of the development process, and solving this issue requires organizational change, which is not covered in this research.

Lastly, some publications don’t mention the problems related to company process or team management. For example, getting permission to try new tools, or deciding who should take care of security, can be difficult. These are important problems, but this study only focuses on technical improvements, not on solving internal business or team issues.

3 RESEARCH METHODOLOGY

This chapter presents the research methodology used to evaluate cybersecurity improvements in e-commerce platform development. It will describe the current security state in one of the services of at Company X, in the research design, data collection methods, data analysis techniques and tools used in the study. The methodology ensures the structured approach to evaluating the security situation of the company X and the impact of implementing automatic dependency updates, security scanning, as well as granular permissions for accesses.

A well-defined methodology is essential to ensure that security enhancement proposed in the study are well implemented and rigorously evaluated. The study

employs a combination of qualitative and quantitative methods, providing realistic evidence of security improvements. By analysing existing vulnerabilities, tracking security automation outcomes, and comparing industry best practices. This study aims to develop a replicable security framework applicable to other e-commerce platforms facing similar challenges.

3.1 Analysis of Current Cyber Security State

Before implementing security enhancements, a current state analysis was conducted to evaluate the existing cyber security situation at company X. This help identify key vulnerabilities, assess the effectiveness of existing security controls, and highlight areas requiring improvement. However, prior to this research, security measures were largely absent with only Single Sign-On (SSO) implemented for a few third-party services. There were no automatic security checks, no structured dependency management, and minimal access control enforcement across internal systems.

Company X operates an extensive e-commerce platform that relies on microservice architecture, third-party integrations with multiple enterprise resource planning (ERP) systems, and cloud-based infrastructure. Security mechanisms currently in place include multiple factor authentication that applied to a few of its services. There were no automated security checks, no structured dependency management, and minimal access control enforcement across internal systems.

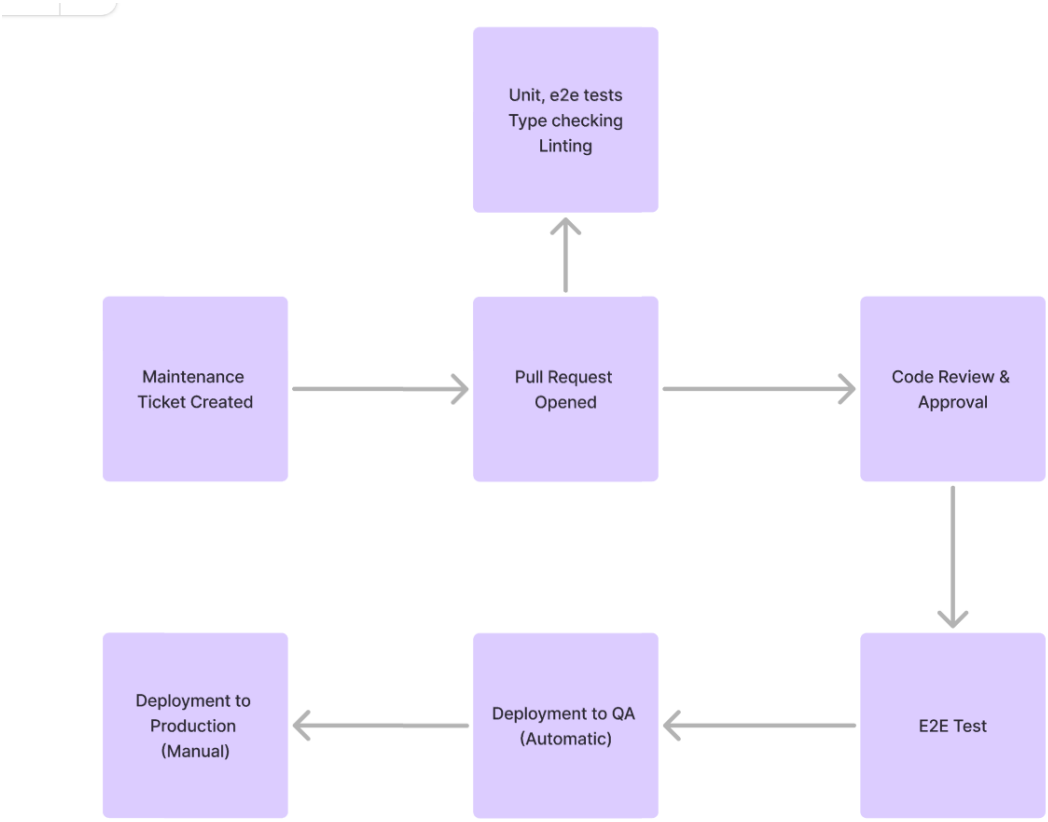

Figure 3.1 Development process

The development process followed by company X involved developers creating Jira tickets for feature implementation or bug fixes, followed by coding, and deployment. However, security considerations were not systematically integrated into this workflow. Code changes were manually reviewed by developers, there were no structured security validations, making it possible for vulnerabilities to go unnoticed. Furthermore, there was not dedicated security team responsible for monitoring vulnerabilities in dependencies or enforcing security policies across microservices.

A security audit of the existing system revealed that the company lacked automated dependency management, leading to outdated and vulnerable third-party libraries. Security scanning tools were not integrated into the CI/CD pipelines, making it difficult to detect and fix vulnerabilities before deployment.

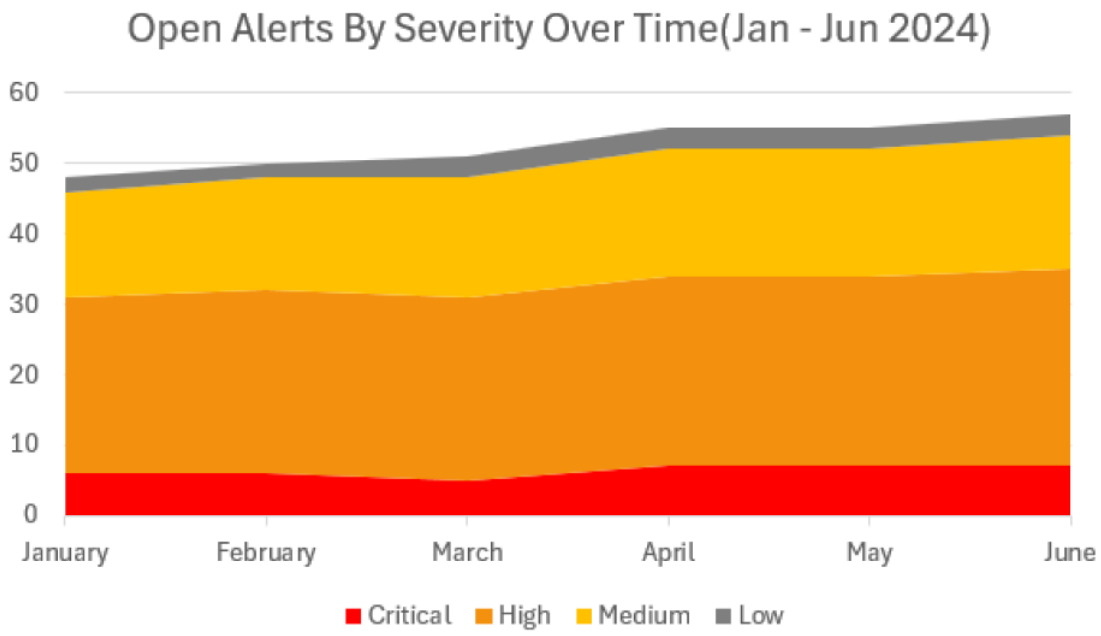

Figure 3.2 Open Alerts by Severity Over Time

Additionally, access control mechanism was inconsistent with, with developers having varying levels of access to critical systems without strict enforcement of the least privilege principles. Without proactive security monitoring, potential threats and vulnerabilities were often discovered late, leaving the system exposed to cyber threats.

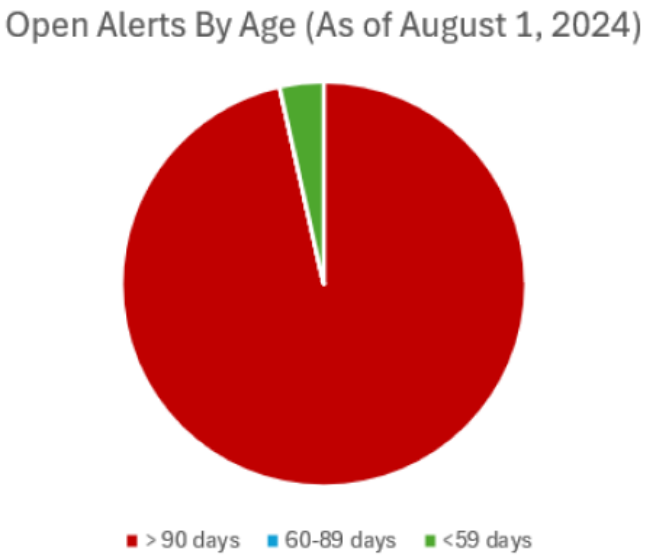

Figure 3.3 Open Alerts by Age

These findings highlighted the need for a structured security framework that integrates automated tools such as GitHub Dependabot for dependency updates, Bearer for security scanning, and stronger access control enforcement mechanisms.

3.2 Study Design

The study employs a case study methodology to analyse security vulnerabilities and assess improvements after implementing security automation. The approach focuses on examining the security challenges faced by the company X before and after applying automated tools and structured security policies. This methodology ensures a real-world application of security best practices and evaluates their impact on mitigating vulnerabilities in an actual e-commerce environment.

The study also includes comparative analysis where the security posture of the company X before security improvements is compared to the result after automation and policy enforcement. This method provides measurable insights into the effectiveness of Github Dependabot for dependency updates, Bearer for code security validation, and the introduction of stricter access controls. Additionally, the research references industry standards such as OWASP Top 10, PCI DSS, and the NIST Cybersecurity Framework to benchmark security practices.

3.3 Data Collection Methods

Data collection for this study includes both primary and secondary sources to ensure a comprehensive analysis of security practices. Primary data is collected through security audits, vulnerability scanning reports, and authentication logs. Static Application Security Testing (SAST) tools such as Bearer, SonarQube are used to scan the source code for vulnerabilities. The study also tracks dependency updates through GitHub Dependabot and examines authentication logs to assess the effectiveness of stricter authentication.

Secondary data is gathered from industry reports, academic publications, and cybersecurity guidelines. Security incident reports from Verizon's Data Breach Investigation Report (DBIR) and research on cybersecurity frameworks such as Zero Trust Architecture and Secure Software Development Life Cycle (SDLC) are reviewed to contextualize findings [10, 9].

3.4 Data Analysis Approach

The study applies both qualitative and quantitative analysis techniques to interpret collected data. With qualitative analysis, there is an involvement of thematic analysis of interview transcripts to identify recurring security concerns and challenges. Additionally, a comparative policy review examines company X's security policies before and after automation, ensuring alignment with industry best practices [2,4]. Quantitative analysis is conducted using descriptive statistics, tracking key security metrics such as the number of vulnerabilities detected before and after automation, frequency of dependency updates, and effectiveness of RBAC enforcement in restricting unauthorized access [5,6].

Regression analysis is used to determine the correlation between security automation adoption and the reduction of vulnerability over time. Performance evaluation measures the efficiency of Github Dependabot, and SAST tools based on detection rates, remediation time, and overall system security. The combination of these analysis approaches provides and evidence-based approach to evaluating cybersecurity improvements at company X [3, 9].

4 ANALYSIS OF THE CURRENT SITUATION

To propose an effective cyber security framework, it is crucial to thoroughly analyse the existing cyber security issues faced by company X. This section presents a detailed evaluation of the microservices used in their e-commerce platform. The analysis is based on data collected from static code analysis tool

and internal system audits. The goal is to identify existing gaps and derive conclusions that form the basis for a secure and scalable framework.

4.1 Microservices inventory and architecture overview

Company X operates over 30 microservices within our e-commerce infrastructure. These services are responsible for core business functions such as payment processing, order management, product management, pricing, and inventory management. Each microservice is developed and maintained independently, often by different teams. While this promotes agility, it also introduces inconsistency in how security is implemented and managed.

An internal audit found that while 100% of the microservices have dependency security configured in their repositories through tools such as GitHub Dependabot, there is a complete lack of follow-up actions. Security alerts are generated but not acted upon, which allows vulnerabilities to remain in production code. This situation implies that developers may not be aware of the alerts, or they lack the process and resources to remediate them.

Most importantly, no services were found to implement Static Application Security Testing (SAST) or Dynamic Security Testing (DAST) as part of their CI/CD. This means that the code is often deployed to production without undergoing automated security validation [5]. As a result, potential vulnerabilities introduced during development are not detected early, and there is no reliable safety net to prevent insecure code from reaching end users. This absence of automated testing mechanisms represents a significant risk to system integrity and customer data safety [3]. Additionally, services differ in their approach to authentication, logging and error handling, making it difficult to maintain a unified security policy across the entire system [4].

4.2 Dependency Health and Vulnerability Statistics

A detailed review of the open vulnerabilities reported by Github Dependabot across ten critical microservices at company X reveals a concerning security posture. These microservices include price related API, sourcing, freight fee calculations, and more. The total number of open alerts is 307, with many remaining unresolved for over a year. This prolonged exposure increases the attack surface and the potential impact in case of exploitation [7]. Figure 4.1 below presents a breakdown of total alert severities found across all own repositories. It clearly shows that high and medium severity issue make up most vulnerabilities, highlighting a critical weakness in the current software supply chain hygiene.

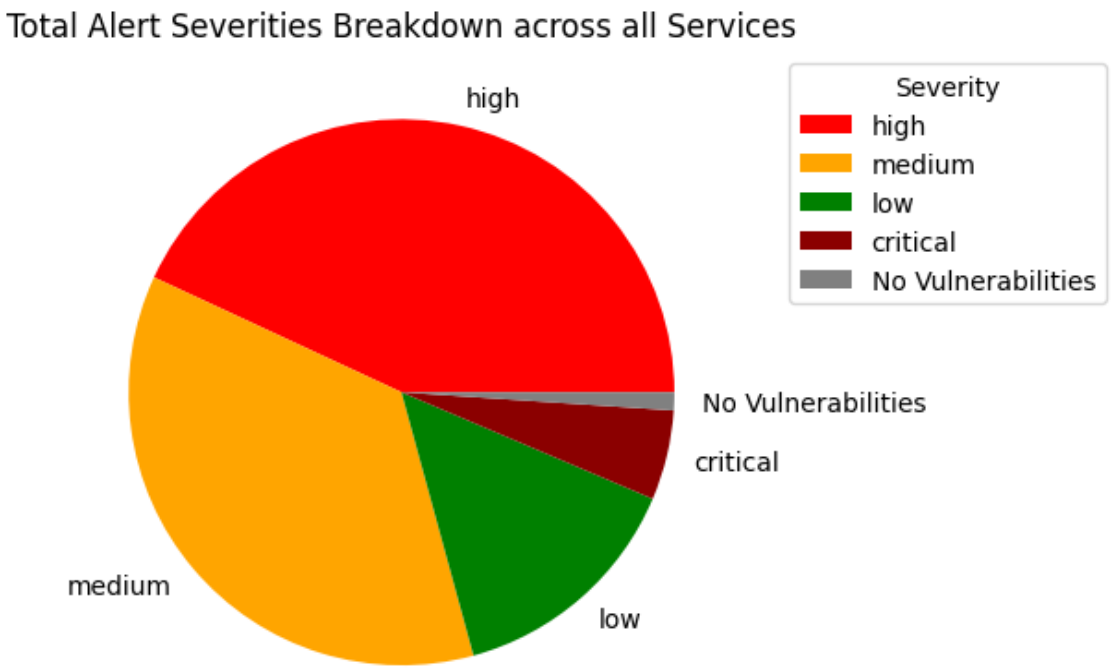

Figure 4.1 Total Severities Breakdown in All Repositories

Figure 4.2 complements this view by showing the distribution of repositories using stack bar charts. It visually compares the relative burden of different severity levels across services, clearly indicating which services are most at risk. For this study, the services have been anonymized.

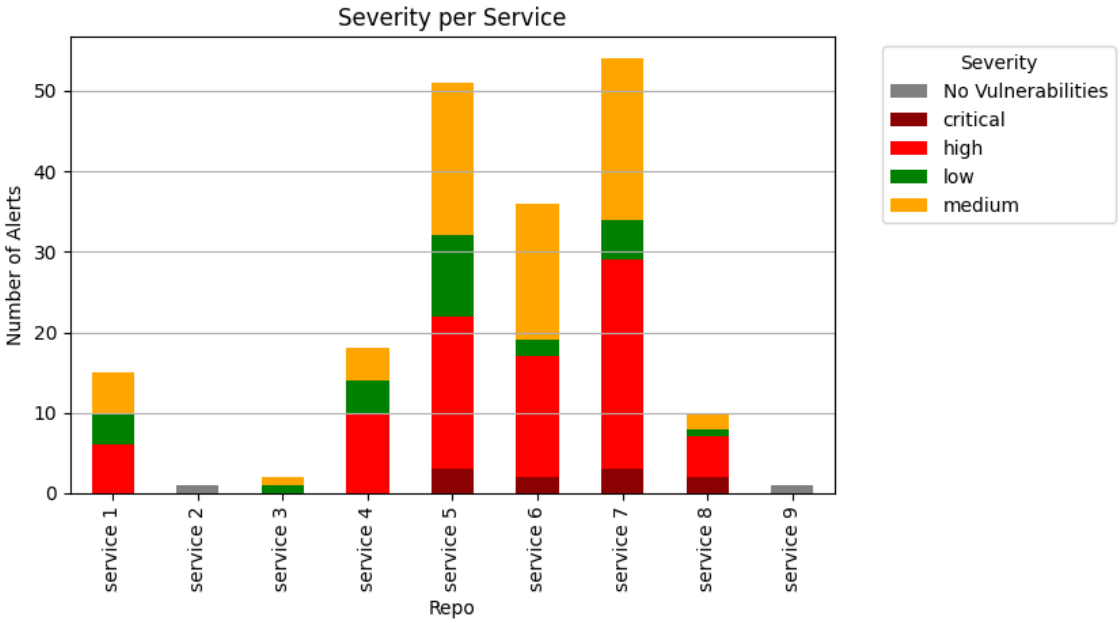

Figure 4.2 Severity per Service

The chart visually compares the relative burden of different severity levels across services, clearing indicating service 5, 6 and 7 are most at risk. It also clearly shows that high and medium severity issues make up most vulnerabilities, highlighting a critical weakness in the current software supply chain hygiene.

Among the analysed services, Service 5 and Service 7 exhibit the highest alert counts, both exceeding 50 known issues. These services also show a disproportionately high number of high and medium severity vulnerabilities, emphasizing their potential risks to the overall system. Service 6 also shows elevated vulnerabilities, particularly in medium and medium severity.

In contrast, service 1, service 2, service 3, and service 9 show significantly fewer alerts with some services even reporting no active reporting no active vulnerability (as shown by the gray bars). This disparity reveals inconsistent security practices across the microservice ecosystem. Despite having security scanning in place, the inaction on alerts poses a major risk for exploitation. The data underscore the necessity for the remediation and the integration of security validation into CI/CD workflows.

The dependency analysis revealed that approximately 77% of the microservices outdated libraries or frameworks. On average, each microservice has at least 25 outdated packages, with some having dependencies that are over two years old and known to contain critical vulnerabilities. These issues are largely due to the absence of an automated dependency management strategy. Developers are not consistently notified of new vulnerabilities or patches, and upgrades are often postponed due to compatibility concerns or limited resources.

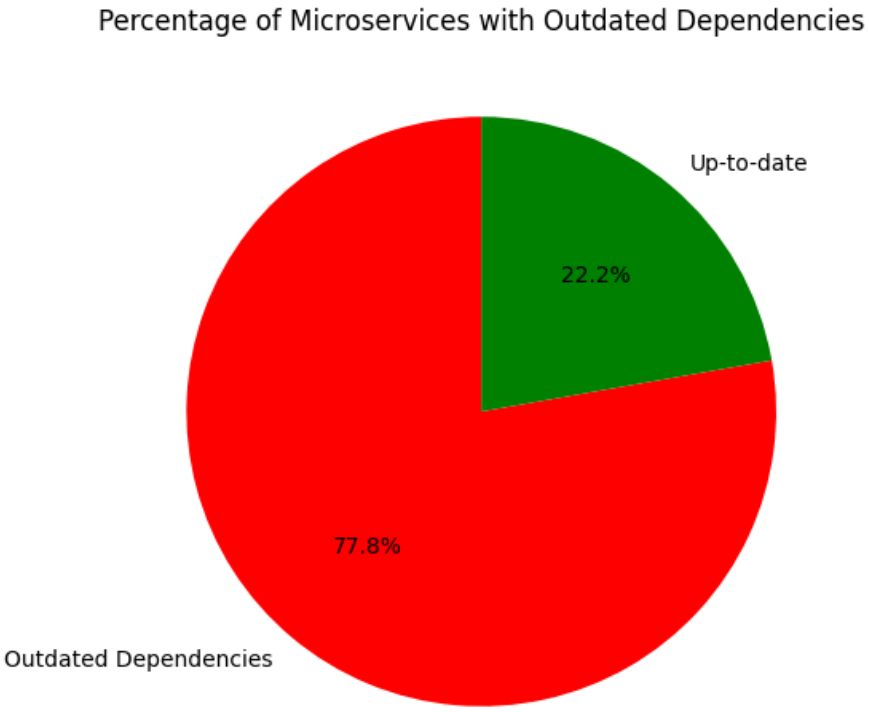

Figure 4.3 Percentage of microservices with Outdated Dependencies

4.3 Code Vulnerabilities and Static Analysis Finding

Static Application Security Testing (SAST) was conducted across a subset of repositories at Company X using Semgrep. The goal of this analysis was to detect insecure coding practices and weaknesses in configuration. The Semgrep scan revealed a total of 16 medium severity findings, with no critical or high severity issues reported in the scanned subset. All findings were linked to insecure

environment variables settings, indicating configurations risks may affect application security under certain conditions [5].

The figure 4.4 below illustrates the number of finding per services. The visualisation highlights that Service 1 had the highest number of findings with five occurrences. Other services like Service 2 and Service 3 had three and two finding respectively, while Service 4, Service 5, and others had one or two.

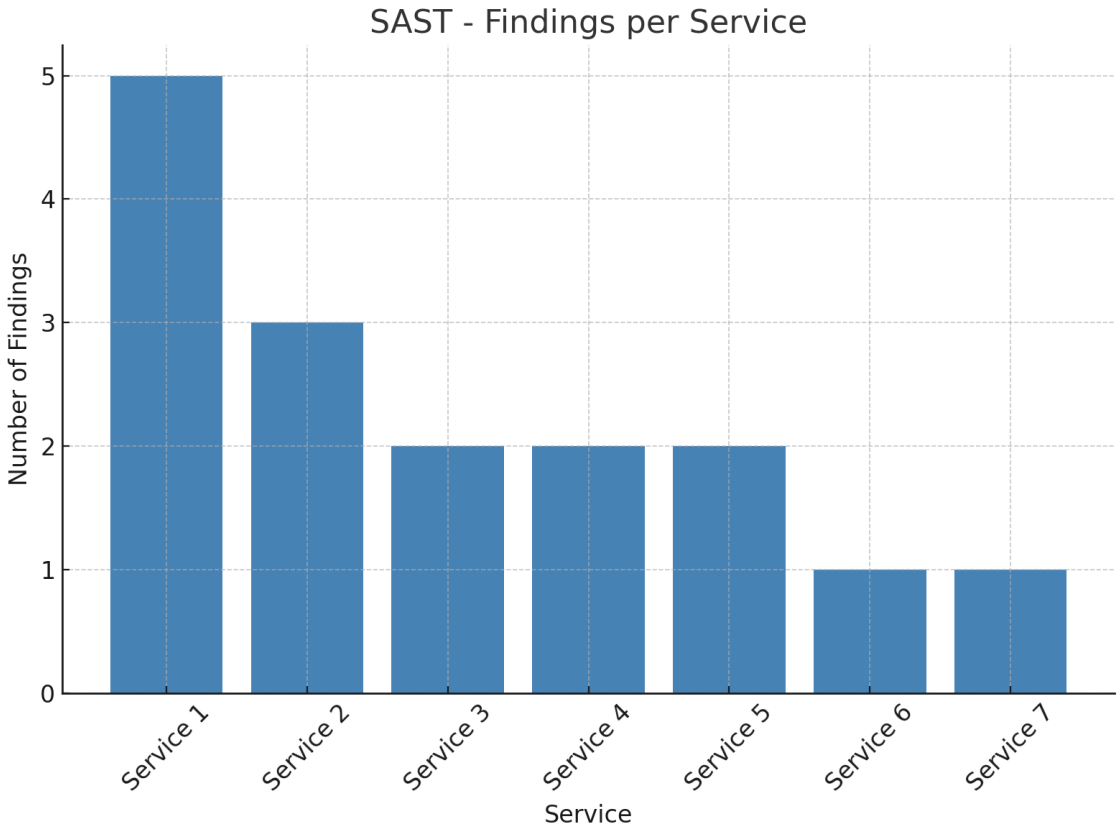

Figure 4.4 SAST Findings per service

The findings suggest a gap in secure coding practices and the absence of static security check during development across the board. Without integrating tools like Semgrep into CI/CD workflows, these misconfigurations can persist into production [5]. Early detection through automated SAST and ongoing security reviews are necessary to strengthen the overall application security posture at the company X. These issues, while not immediately critical, indicate a lack of

early automated testing and adherence to secure coding practices. Integrating SAST into CI/CD pipelines and improving developer awareness are necessary next steps to avoid these types of misconfigurations and improve code quality across all microservices in the platform solution [8].

4.4 Access Control and Security Principles Implementation

An internal audit of access control across 20 services used by company X revealed inconsistencies in implementing identity and access management practices. Notably only 9 out of 20 services had Single Sign-on enabled, while 2 services granted admin access to all users, violating the principle of least privilege. In terms of role-based access control, 14 services had it properly configured, while the remaining 6 lacked role-specific enforcement. Figure 4.5 presents a bar chart summarizing these key findings, highlighting that critical access mechanisms are not uniformly applied across the organization.

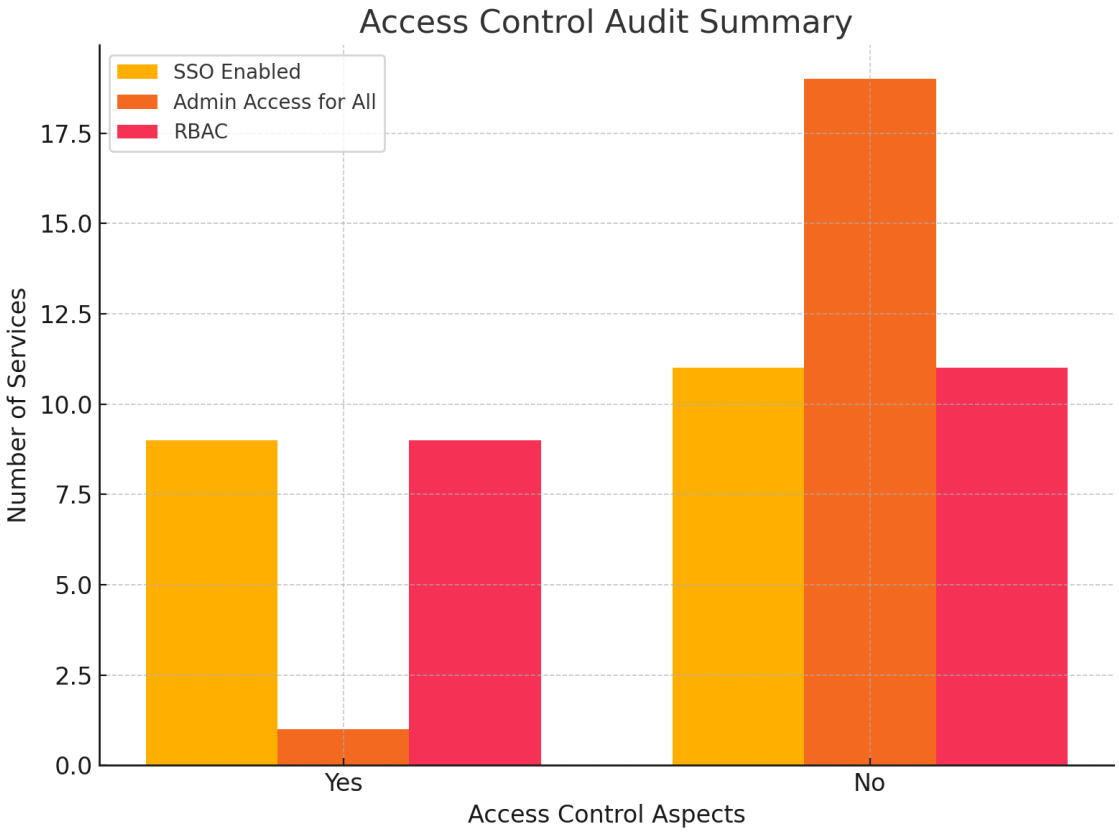

Figure 4.5 Access Control Audit Summary

In addition to access control, the audit also evaluates whether services adhered to the Confidentiality, Integrity, and Availability principles. The results showed that while all services uphold confidentiality, 8 services lacked integrity enforcement and 9 did not ensure high availability. These gaps present risks ranging from data tampering to system outages.

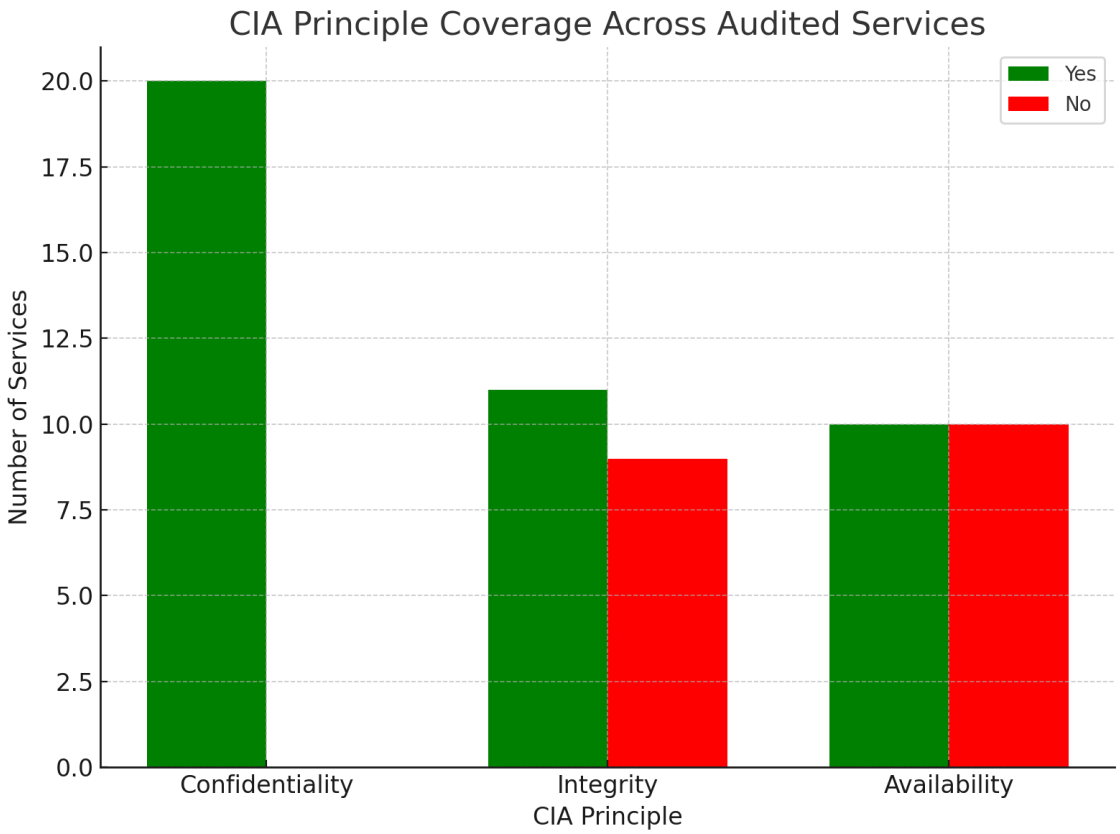

Figure 4.6 CIA principles coverage across audited services

The failure of enforcing role-based access control consistently and over-privilege of users increased the risk of unauthorized access and data manipulation [3]. Moreover, missing controls over audit trails and change logs pose an insider thread. These deficiencies reflect not only a lack of centralized governance but also highlight the basic security hygiene. The fill this gap, company X must implement organization-wide SSO, adopt strict RBAC models, and enforce access policies based on job roles and operational need.

4.5 Summary and key findings.

The analysis clearly demonstrates that company X's current security setup lacks standardization, automation, and proactive controls. From the inconsistent security scanning of microservices to the widespread presence of outdated dependencies and unaddressed vulnerabilities, the system presents multiple

entry points for potential attacks. Additionally, the lack of integrated SAST tools and inadequate application of secure coding principles has resulted in many code-level weaknesses.

Moreover, the current access control measures are not strong enough to protect critical systems. Role-based access control is applied inconsistently, and many developers have been granted excessive permissions, which creates a serious risk of internal threats. The audit also revealed a lack of centralized governance and limited security training, which leads to uneven enforcement of key principles like least privilege across different services.

The key takeaways include:

- Widespread use of outdated and vulnerable dependencies
- Lack of consistent static code scanning and forcing of security standards
- Weak access control mechanism and insufficient role-based access policies
- Fragmented implementation of security principles across services

These findings together form a compelling case for the implementation of a structured security framework that embeds security into the development lifecycle and standardizes practices across services.

5 PROPOSED CYBER SECURITY FRAMEWORK FOR E-COMMERCE PLATFORM DEVELOPMENT

The cyber security framework proposed in this thesis is founded on a set of principles that align with best practices in secure software development, particularly in the context of microservice-based ecommerce platforms. These principles guide the structure, objectives, and implementation of strategy of the framework to ensure that it remains both effective and adaptable.

Firstly, the principles of Security by Design [source] are crucial to the framework. Security is not treated as an isolated concern at the end of development, but rather as integrated element from the earliest phases of system design. This stand supports the identification and mitigation of security risks before they can escalate into vulnerabilities. The framework also adheres to the concept of Shift-Left Security, which promotes the early incorporation of security testing and controls in the software development lifecycle [6]. This approach not only reduces the cost of addressing issues but also foster a culture of security awareness among development teams. The Principle of Least Privilege further underpins the access control mechanisms within the framework. Each user and system component are granted the minimum level of access required to perform its intended function [3]. This principle is closely aligned with “Zero Trust” [source] security model, which assumes that no users or system should be trusted by default. All interactions are subject to verification, authorization, and continuous monitoring [7]. Another important aspect is the Automation First philosophy. Given the dynamic nature of modern software development, the framework encourages the use of automated tools for tasks such as dependency updates, static code analysis, and security policy enforcement. Automation reduces the risk of human error and increases the consistency and scalability of security practices [5, 9].

Furthermore, the framework is designed to be modular and scalable. It supports different levels of system complexity, making it suitable for small- and large-scale e-commerce development environments. This modularity ensures that security measures can be incrementally adopted and adapted to without disrupting existing workflows. Finally, the framework is tailored to the specific needs of the ecommerce sector. This includes requirements for protecting customer data, managing transactions and complying with relevant regulatory requirements such as GDPR. The framework addressed these requirements by embedding best practices that support confidentiality, integrity and availability across all services.

The proposed framework is composed of five key pillars:

1. Automated dependency management
2. Static Application Security Testing
3. Role Based Access Control and Identity Governance
4. CIA Principal Alignment
5. Continuous Monitoring and Feedback Loop

One of the most significant risks identified in the current environment was the widespread use of outdated libraries with known vulnerabilities. To mitigate this, the framework proposes the integration of automated dependency management tools within each microservice repository. Tools such as GitHub Dependabot, Renovate, and Snyk are suitable for this purpose and can be configured to scan and alert on insecure and outdated packages on a scheduled basis [7]. This element of the framework ensure that dependencies are regularly scanned, and updates are automatically suggested to developers via pull requests. Security thresholds can be defined to prioritize critical and high-risk vulnerabilities. Furthermore, ownership and triage processes should be clearly assigned to development teams to ensure that remediation actions are taken in a timely manner. The objective is to reduce the manual overhead and improve the visibility of risks within the software supply chain [4].

The absence of structured SAST procedures across company X's services presents a serious threat, allowing insecure code to reach production. To resolve this, the framework includes the integration of tools such as Semgrep, SonarQube, or CodeQL into CI/CD pipelines. These tools can identify common coding errors, insecure configurations, and vulnerabilities defined by standards like OWASP Top 10 [6]. SAST tools should be configured to run automatically during pull request checks or before merging to the main branch. Developers receive immediate feedback on insecure patterns, enabling them to address issues early in the software lifecycle. This approach supports the philosophy of Shift Left Security, ensuring that vulnerabilities are discovered and mitigated during development rather than post-development [5].

The framework emphasizes the enforcement of RBAC to mitigate risks associated with overly permissive access. In the current state, multiple services allowed all users administrative privileges, which contradicts the principles of least privilege [3]. The proposed framework mandates that every user, developer, and third-party integration must be assigned only the minimum permissions necessary to perform their job functions. In addition, centralized authentication via SSO should be adopted using providers such as Azure AD or Auth0. Access policies must be governed by roles and review periodically. Multi-factor authentication (MFA) should be enforced for all privileged accounts. This policy ensures traceability reduces the risks of account misuse and aligns with the enterprise security standards [7].

The CIA triad forms the foundation of security governance. This framework ensures that all microservices explicitly implement practices that uphold these principles. For confidentiality, all sensitive data must be encrypted at rest and in transit using modern cryptographic standards. For integrity, services must implement robust validation logic, audit trails, and tamper detection mechanisms. For availability, the framework recommends redundant infrastructure, service health checks, and automated recovery procedures to minimize downtime. Each service owner is responsible for documenting how their service upholds the CIA principles. These implementations should be reviewed during security audits and as part of on-going assessments [4].

Cyber security is not a one-time initiative but an ongoing process. This component of the framework introduces continuous monitoring of services and feedback mechanisms to adapt to emerging threats. Logging alerting, and dashboarding are essential for visibility and incident detection. Key metrics such as number of open vulnerabilities, frequency of dependency updates, and access control violations should be tracked [5]. Dashboards can be built using platforms such as Dynatrace or Datadog. Monthly reviews should be conducted to evaluate trends, refine policies, and drive continuous improvement [9].

To better illustrate the structure of the components of the proposed cybersecurity framework, a conceptual mind map in the figure 5.1 has been developed to visualize the interconnection between its five core pillars and their supporting practices. Each block represents one main component that was discussed earlier.

First Automatic Dependency Management focuses on reducing risks from outdated third-party dependencies. It enables automated updates, provides security alerts, and enforces dependencies to make sure that packages with known vulnerabilities are updated. On the other side, Static Application Security Testing (SAST) deals with finding security issues in the code itself with tools such as Semgrep or SonarQube to scan the codebase and applying security rules and tracking issues before the code is released to production.

Below the first two components, there are three more components. Role-based Access Control (RBAC) and Identify Governance ensures that users only have access to the parts of the system that they really need. It includes role assessments, access policies, and Single Sign On (SSO). Next, CIA Principal Alignment makes sure that all services in the platform respect the principles of Confidentiality, Integrity, Availability, which is the foundation of a secure system. Lastly Continuous Monitoring and Feedback Loop provides real-time visibility into security performance through security logs, alerts and risk assessments. It helps organization detect issues early and improve security policies based on ongoing feedback.

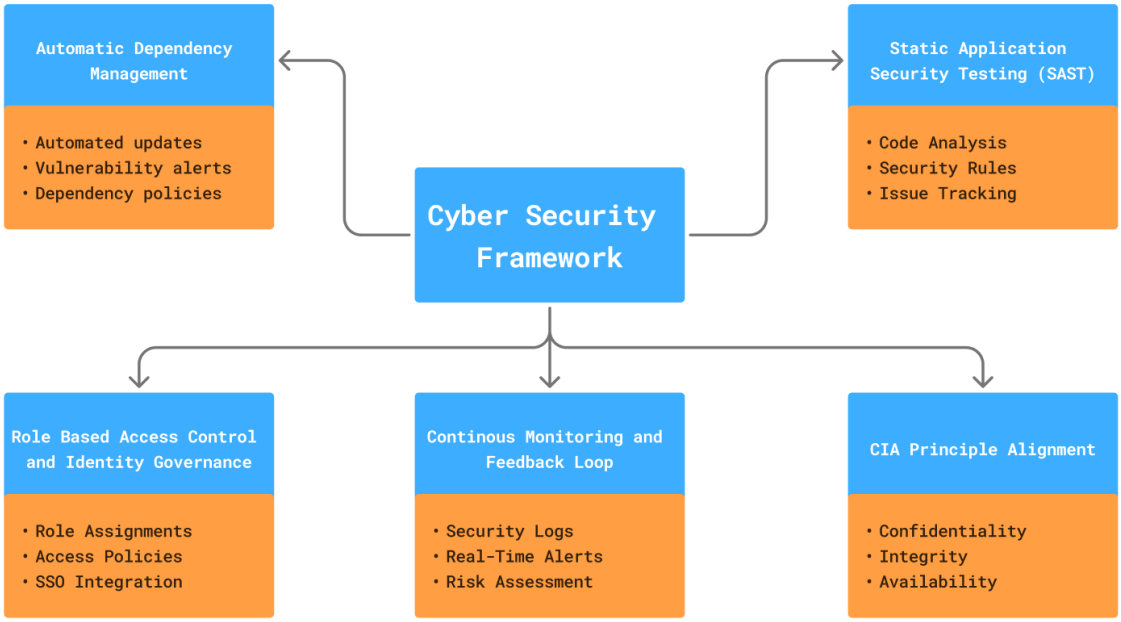

Figure 5.1 Proposed Cyber Security Framework

The proposed framework provides a blueprint for embedding security practices within the software lifecycle of e-commerce microservices. In the following section, this framework will be applied in a case study to evaluate its practical implications and performance in a real-world environment.

Initially, teams should focus on Automated Dependency Management and Static Application Security Testing, as these can be integrated into existing CI/CD pipelines with relatively low overhead. These areas address immediate risks related to outdated libraries and insecure coding patterns [5]. Once basic automatic and security scanning are set up properly, the next phase involves implementing Role Based Access Control (RBAC) and Identity Governance. This phase requires role definitions but provides significant values by reducing insider threats and enforcing access policies [3]. Following access governance, the organization should work toward the full CIA Principles Alignment. This involves verifying encryption standards, integrity controls, and service availability practices across all systems. This phase may require more technical planning and infrastructure support [6]. Finally, the organization should establish Continuous Monitoring and Feedback Loops. This step reinforces earlier implementations by

providing visibility, alerting, and iterative policy improvements. Dashboards and automated alerts support long-term sustainability of security posters [9].

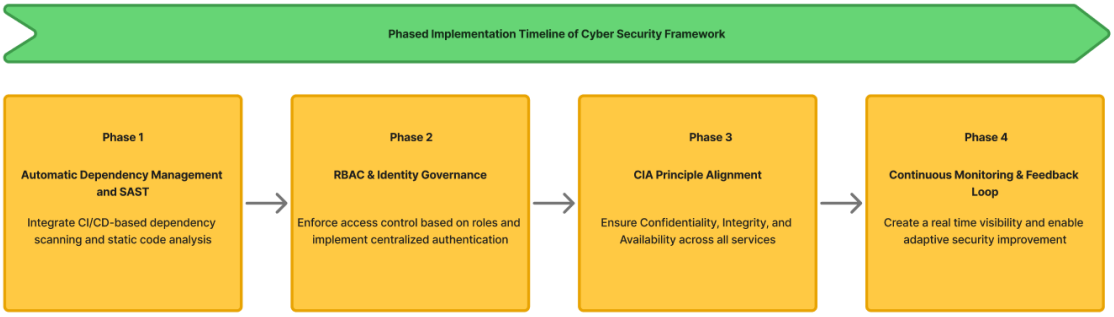

Figure 5.2. Phased Implementation Timeline of Cyber Security Framework

6 EVALUATION AND APPLICATION OF THE POTENTIAL OF THE FRAMEWORK

This chapter aims to evaluate the applicability of the proposed cybersecurity framework in a realistic development setting, especially one based on a microservice oriented e-commerce platform such as that of company X. Due to time and operational constraints, a fully implemented case study was not carried out. Instead, a scenario-based walkthrough approach was adopted, focusing on how the framework could be integrated into the existing workflows and technical architecture. This method highlights practical considerations, anticipated outcomes, and potential challenges associated with each pillar of the framework.

In dependency management, the integration of automated dependency tools like GitHub Dependabot begins by addressing the existing backlog of security vulnerabilities manually. Once a stable baseline is achieved, Dependabot is configured to scan repositories twice a week. When issues are detected, it automatically generates pull requests (PR). These PRs are then subject to the usual CI pipeline, which includes unit testing, reviewer assignments, and Slack notifications to the relevant teams. Developers are expected to review and merge PRs if the associated tests pass. For updates that introduce errors or break functionality, teams either revert changes or address them manually. In situations where testing is insufficient or updates are too disruptive, issues are logged in

Jira and prioritized accordingly [5]. Initially, the increase in open PRs may add overhead, particularly for services with legacy dependencies. However, over time, the system matures, and the frequency of updates and risks decreases. To facilitate adoption, documentation is maintained in Confluence and team wide accountability is supported through quarterly security goals. Regular team check-ins help ensure steady progress. Additionally, custom configurations allow teams to manage update frequency, reducing the likelihood of unexpected service disruptions. The long-term goal is to create a dependable, low-friction update mechanism that increases codebase reliability and minimizes manual intervention [6].

Incorporating SAST tools such as Semgrep into the CI/CD pipeline enhances the ability to catch vulnerabilities early in the development lifecycle [5]. In the proposed setup, scans are run automatically at the pull request and merge stages. Vulnerabilities with a severity of medium or above cause the CI to fail, ensuring that problematic code does not proceed to production. Findings are posted as inline comments within the pull request, giving developers immediate visibility into which parts of the code triggered the alert. Developers are responsible for addressing the issues, and fixes must be reviewed by another team member to avoid bias [7]. In cases of false positives, developers may bypass the rule but must document their reasoning in the configuration file. While no formal training exists yet, a central knowledge base is proposed to help teams understand how to interpret results and maintain the configuration. Looking forward, it is recommended to track metrics like average time to resolve SAST issues, the number of repeated rule violations, and the ratio of true to false positives. These metrics would support continuous improvement in secure development practices [8].

Company X uses Azure AD as its identity provider and has implemented SSO for most services. However, the application of RBAC is inconsistent, and several services continue to grant administrative access by default. An internal review showed that while roles such as Developer, Editor, Viewer, and Admin are well-defined and documented, there is no central system for managing or auditing role

changes. Access rights are manually reviewed by two administrators monthly. Still, the lack of automated governance has resulted in past incidents, including the accidental revocation of a production token due to unrestricted access. Addressing this requires adopting stricter access boundaries and integrating permission audits into operational routines. To balance flexibility and control, temporary permission escalation models such as just-in-time access for on-call developers can be introduced. Additionally, regular access reviews and integration of RBAC into onboarding and offboarding workflows would reduce risk while maintaining developer efficiency. Implementing these measures would significantly strengthen internal controls and minimize the attack surface from within [4].

Company X has already taken several steps toward aligning with the CIA triad. For Confidentiality, all services operate over HTTPS, and credentials are stored securely using environment variables or secrets management systems. Access to customer data is also controlled via service-level authentication policies. However, Integrity remains an area of improvement. While services implement strong input validation and some audit logging, there are no safeguards like file checksums or immutable logs. Adding cryptographic hash checks or adopting write-once logging mechanisms would enhance traceability and protection against tampering [4]. In terms of Availability, basic health monitoring and restart procedures are in place, and the recovery process is documented. Still, redundancy across regions or zones has not been implemented, making services vulnerable to regional outages. Introducing distributed deployments and automated failover mechanisms would improve fault tolerance [5]. Currently, CIA principles are not formally tracked, and teams are not always aware of their responsibilities in this area. To improve awareness, it is suggested to create a lightweight checklist per service for monthly self-evaluation, gradually embedding CIA accountability into the development process [3].

Dynatrace is used as the primary monitoring platform, but it currently lacks dedicated security metrics. In the envisioned state, dashboards would include KPIs such as unresolved dependency alerts, frequency of SAST violations, and

audit trail anomalies. This would provide real-time insight into the organization’s security posture [5]. Security incidents are already addressed through Jira and discussed in weekly meetings, but a more structured monthly review process is recommended. This review would include dependency health, access controls, SAST metrics, and CIA checklists. Gradually, the goal is to transition from reactive discussions to proactive risk management. Although some components of the framework are already in partial use, this walkthrough shows how each pillar can be strengthened and integrated within existing workflows.

By adopting the framework incrementally, organizations can significantly enhance their security posture without disrupting their development operations [2]. The recommendations presented here reflect realistic steps that align with current industry practices and accommodate organizational constraints [7].

7 CONCLUSION AND DISCUSSION

This thesis began with the intention of understanding and addressing real world cybersecurity challenges in the development of microservice-based ecommerce systems, using company X as a practical case. Throughout the study, I explored the existing practices in use across various services, identified recurring vulnerabilities, and proposed a set of improvements that are both realistic and aligned with current developments workflows. The result was a structured framework made of five parts: automated dependency management, static application security testing, role-based access control with identity governance, CIA principal alignment, and continuous monitoring. These elements are not entirely new on their own, but their value comes from how they are combined and adapted to a phased model that suits the environment I studied.

The scenario-based walkthrough played a central role in shaping the direction of the proposed framework. Rather than relying solely on theoretical concepts, the focus was to place on how these security improvements could be applied in real, every day development processes. Throughout the project, it became clear that while many tools and methodologies are available, they are often underutilised due to limited time, lack of awareness, or absence of clear ownership. This observation highlighted that technical solutions alone are not sufficient, but organisational habits and workflows must also evolve to support secure development practices.

One of the most significant findings during this study was the widespread presence of outdated dependencies and minimal effort devoted to addressing them. This was not only a technical gap but also reflected a broader issue in security culture and mindset. As the work progressed, I began to observe more security related concerns during code reviews and development planning with issues that I may have overlooked previously. This shift in awareness and attention to potential risks represents a key personal learning outcome of the thesis.

Due to the time limitation, it was not possible to fully implement and validate the framework in a live production environment. If more time had been available, the focus would have included a clearer rollout plan for each phase and the collection of feedback from internal security professionals. These steps would have contributed to further refining the framework and tailoring it more precisely to different teams and service areas. The scale of the system, with its many independently managed microservices, presented challenges in applying a single unified approach. Despite these limitations, the proposed framework can still provide practical guidance for other development teams, especially those facing similar issues with fragmented or reactive security processes. It brings together familiar tools and concepts into a more structured, actionable format, making it easier for teams to improve their overall security posture in a manageable way.

To conclude, this thesis has laid out a framework that connects technical best practices with realistic implementation steps. While not everything could be tested in full, the ideas and structure provide a strong starting point for improving cybersecurity in ecommerce platforms that rely on microservices.

References

1. Liu X, Ahmad SF, Anser MK, Ke J, Irshad M, Ul-Haq J, et al. Cyber security threats: A never-ending challenge for e-commerce. Frontiers in Psychology. 2022 Oct 19;13.
2. Badotra S, Sundas A. A systematic review on security of E-commerce systems. International Journal of Applied Science and Engineering. 2021 Jun 1;18(2):1–19.
3. Oguta GC. Securing the virtual marketplace: Navigating the landscape of security and privacy challenges in E-Commerce. GSC Advanced Research and Reviews. 2024 Jan 14;18(1):084–117.
4. Kuruwitaarachchi N, Abeygunawardena PKW, Rupasingha L, Udara SWI. A Systematic Review of Security in Electronic Commerce- Threats and Frameworks. Global Journal of Computer Science and Technology. 2019 Feb 7;33–9.
5. Radhika K, Sundar G, Kumar S, Srinivasan TN. A Study of Cyber Security Challenges and its Emerging Trends on Latest Technologies. International Journal of Innovative Research in Advanced Engineering. 2023 Jun 23;10(06):379–82.
6. Admass WS, Munaye YY, Diro A. Cyber security: State of the art, challenges and future directions. Cyber Security and Applications. 2023 Oct 6;2:100031–100031.
7. Safitra MF, Lubis M, Fakhrurroja H. Counterattacking Cyber Threats: A Framework for the Future of Cybersecurity. Sustainability. 2023 Sep 6;15(18):13369–13369.
8. Rajasekharaiah KM, Dule CS, Sudarshan ECG. Cyber Security Challenges and its Emerging Trends on Latest Technologies. In: IOP Conference Series Materials Science and Engineering. 2020. p. 022062–022062.
9. Ukwandu E, Farah MAB, Hindy H, Brosset D, Kavallieros D, Atkinson R, et al. A Review of Cyber-Ranges and Test-Beds: Current and Future Trends. Sensors. 2020 Dec 13;20(24):7148–7148.

10. Verizon. 2023 Data Breach Investigations Report. Verizon; 2023. Available from: https://www.verizon.com/business/resources/reports/dbir/
11. IBM. 2024 Cost of Data Breach. Available from: Cost of a data breach 2024 | IBM