TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Karoliina Koppel 182512

Securing Software Supply-Chain using OWASP Application Security Verification Standard: A SimplBooks Case Study

Master's thesis

Supervisor: Toomas Lepik
Master of science

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karoliina Koppel 182512

Tarkvara arendusahela turvamine kasutades OWASP rakenduse turvalisuse verifitseerimise standardit SimplBooks näitel

Magistritöö

Juhendaja: Toomas Lepik
Magister

Tallinn 2021

Author's declaration of originality

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

Author: Karoliina Koppel

14.05.2021

Abstract

In the last decade software supply-chain attacks have become more frequent and impactful. It is crucial to secure software against possible attacks. The aim of this thesis was to determine, if using an application security standard provides enough protection against historically more frequent software supply-chain attacks.

The thesis is conducted as a case study by analysing the impact of integrating the selected software security standard, OWASP Application Security Verification Standard, on the SimplBooks accounting software. The results show that using the selected standard is helpful against most types of common software supply-chain attacks but lacks the requirements for training the employees about the human factors in cybersecurity.

This thesis is written in English and is 64 pages long, including 4 chapters, 17 figures and 3 tables.

Annotatsioon

Tarkvara arendusahela turvamine kasutades OWASP rakenduse turvalisuse verifitseerimise standardit SimplBooks näitel

Viimase kümnendi jooksul on tarkvara tarneahela rünnakud muutunud aina sagedasemaks ja mõjusamaks. Tarkvara võimalike rünnakute eest kaitsmine on muutunud väga oluliseks. Käesoleva lõputöö eesmärgiks oli välja selgitada, kas rakenduse turvastandardi kasutamine taga piisava kaitse ajalooliselt sagedasemate tarkvara tarneahela rünnakute eest.

Lõputöö viidi läbi juhtumiuuringuna, analüüsides valitud tarkvara turvastandardi, OWASP rakenduse turvalisuse verifitseerimise standardi mõju raamatuidamistarkvarale SimplBooks. Lõputöö tulemustest saab järeldada, et valitud turvastandard on kasulik enamiku levinud tarkvara tarneahela rünnakute korral, kuid selles puuduvad nõuded töötajate koolitamiseks küberturbe inimtegurite teemal.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 64 leheküljel, 4 peatükki, 17 joonist, 3 tabelit.

List of abbreviations and terms

ASVS	Application Security Verification Standard
CAWE	Common Architectural Weakness Enumeration
CIS	Center for Internet Security
CSRF	Cross Site Request Forgery
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DAST	Dynamic Analysis Security Testing
FBI	Federal Bureau of Investigation
IoT	Internet of Things
ISMS	Information Security Management System
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NVD	National Vulnerability Database
OS	Operation System
OWASP	Open Web Application Security Project
SAST	Static Application Security Testing
SDL	Security Development Lifecycle
SDLC	Software Development Lifecycle
SP	Special Publication
UI	User Interface
UX	User Experience
XSS	Cross-Site Scripting
XXE	XML External Entities

Table of contents

List of figures

List of tables

Introduction

As of 2021, software supply-chain attacks have become more and more prevalent. In a report published in March 2021, the Atlantic Council's Cyber Statecraft Initiative found that software supply-chain intrusions have become more frequent and impactful over the last decade [1]. It is essential to focus on securing the development process of the software to prevent software supply-chain attacks. Using a software security standard could be a way for companies to assure the security of their product.

In this thesis, I examine how applying the Open Web Application Security Project's (OWASP) Application Security Verification Standard (ASVS) can help to protect the software development lifecycle against software supply-chain attacks. The main question of this thesis is to understand if the security of the historically more vulnerable stages in a software development process can be improved by introducing a security standard into the development process.

I will examine this hypothesis in the real-world setting by applying the ASVS to SimplBooks, a web-based accounting software. I will analyse if the security of the supply chain can be enhanced by introducing this standard.

The thesis is conducted as a case study to investigate the applicability of the ASVS in a real-world setting and see if the security of the SimplBooks software supply chain can be enhanced by introducing this standard.

In the first chapter of the thesis, a literature review examines the software supply chain and different development models, different application security standards, and multiple known software supply-chain attack cases discovered in the last decade.

In the second chapter, the case study methodology is used to audit the SimplBooks software using the ASVS document, and improvements for the development process are made and suggested.

The third chapter examines if the application of the ASVS to the SimplBooks software fulfilled the purpose of securing the development process against historically frequent software-supply chain attacks. In the fourth chapter, there is a discussion about the shortcomings of the used standard and other possible selections.

1 Literature review

1.1 Secure development lifecycle

1.1.1 Software development lifecycle models

Software Development Lifecycle (SDLC) is a framework that considers the structure of an application from its initial feasibility study through to its implementation and maintenance [2]. SDLC models are used to describe the steps that are taken within the lifecycle framework (Figure 1). Different models are used for different cases. It is essential to determine the model based on what type of software is being developed and what is its functionality.

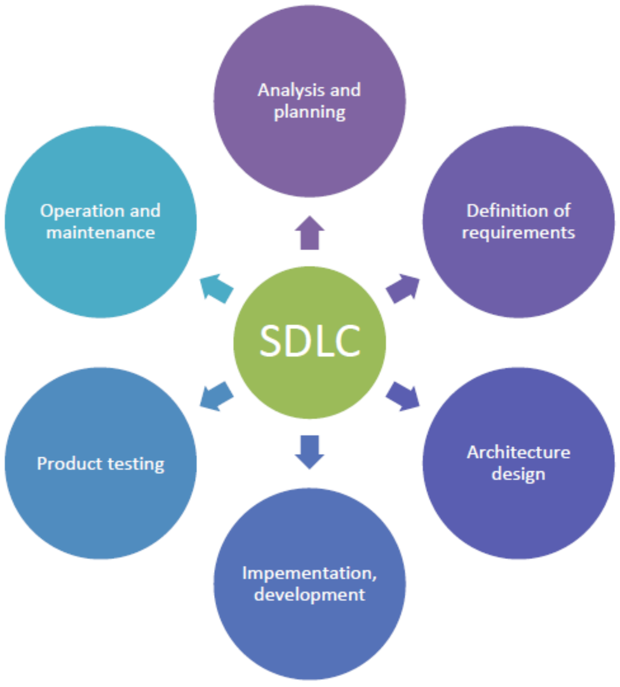

Analysis and planning
Definition of requirements
Architecture design
Impementation, development
Product testing
Operation and maintenance
SDLC

Figure 1 Software development life cycle [3]

Waterfall Model

The waterfall model, also known as the cascade model, is the oldest type of SDLC model, first documented in 1956. The waterfall model recommends that the software be developed in multiple steps. It is recommended that the previous step should be completed before the next one begins. The modified version of this model (Figure 2) in 1970 by Winston Royce also recommends revisiting the preceding step by providing feedback [4]. One step is split into two parts: one part performs the task of the step whilst the other verifies the result. The waterfall model is most efficiently used for software that provides back-end functionality, typically software that provides service to other applications [2]. The waterfall model is not particularly useful if the requirements are dynamic [5]. B-model and incremental model are both modifications of the waterfall model that add iterations [2].

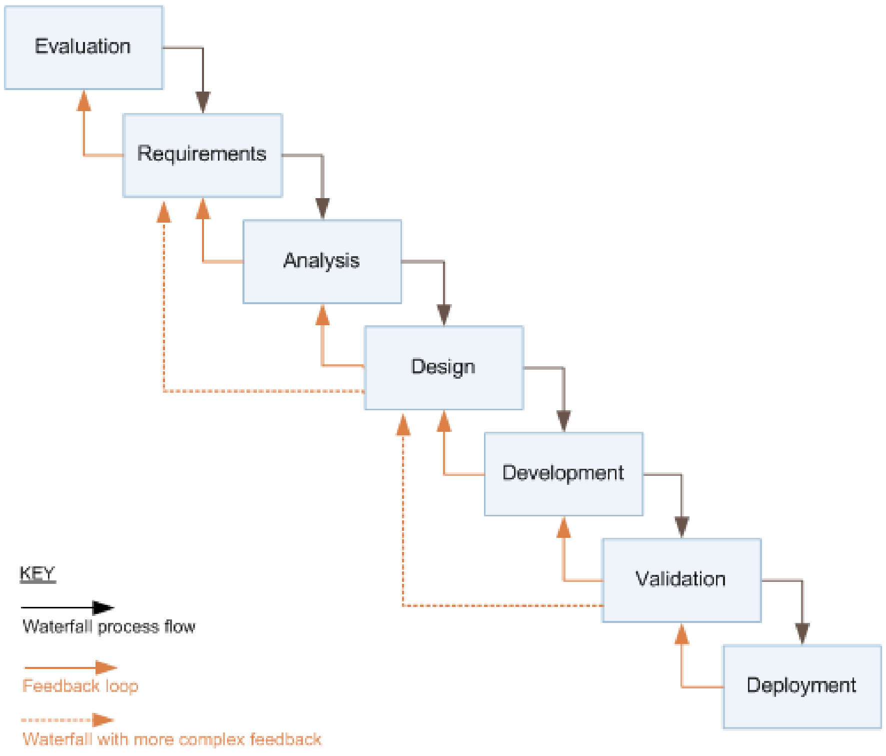
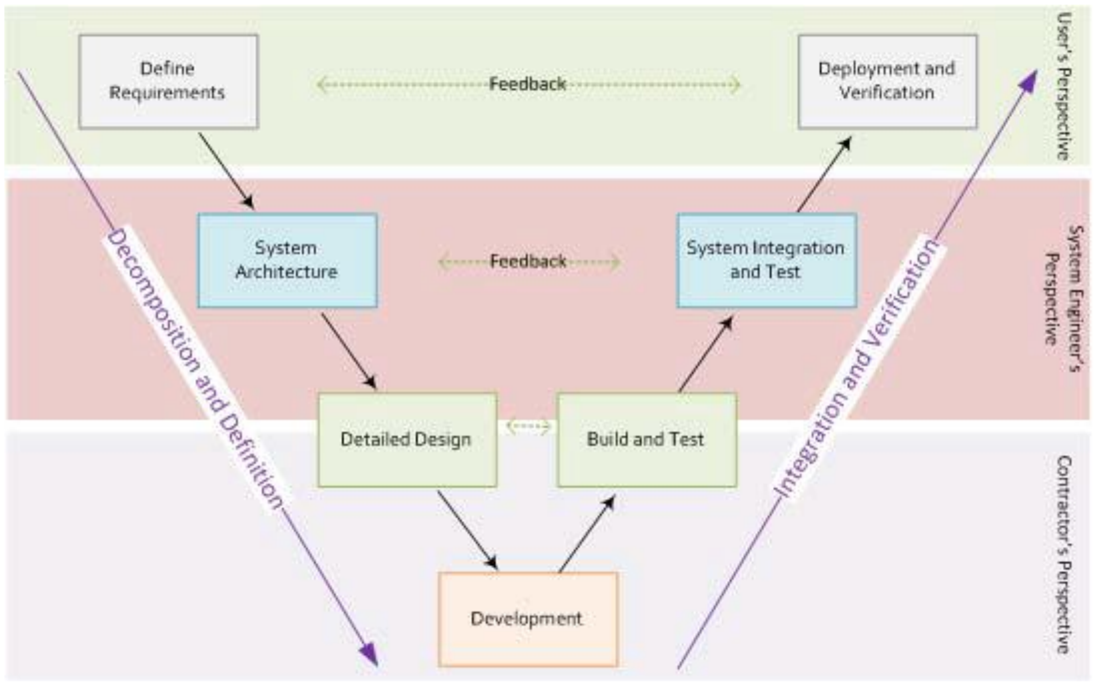


Figure 2 Waterfall model with Royce's interactive feedback [2]

V-Model

NASA developed V-model in 1991 [6]. The V-model is a variation of the waterfall model in a V shape folded in half the lowest level of decomposition. The left leg of the shape represents the evolution of user requirements, and the right leg the integration and verification of the system components (Figure 3). The V-model is symmetrical across two legs so that the right leg can be verified against the corresponding stage of the left leg. Like the waterfall model, the v-model is only helpful for development if the user requirements stay static [2].



Define Requirements
Feedback
Deployment and Verification
User's Perspective
System Architecture
Feedback
System Integration and Test
System Engineer's Perspective
Detailed Design
Build and Test
Development
Contractor's Perspective
Decomposition and Definition
Integration and Verification

Figure 3 V-model [2]

Spiral Model

The spiral model is also a modification of the waterfall model [2]. In this model, a prototype is built in every cycle, verified against the system requirements and validated through testing. Each cycle is an evolutionary development of the last cycle. In every cycle, the risk is analysed and managed. Review is always the last step of each cycle. The main benefit of using this model is that it attempts to contain project risks and costs at the inception. The spiral model is much more flexible for other types of software, unlike the regular waterfall model.

Rapid Application Development

Rapid Application Development (RAD) is a methodology that use prototyping is a mechanism for iterative development [2]. It promotes a collaborative environment where everyone participates actively in prototyping and testing.

Agile

Agile development is used when development occurs in short intervals, and software releases are made to capture small incremental changes [2]. In agile development, the project is broken into small sub-projects. Core agile principles are: working on software is more important than documentation; interactions and individuals are more important than processes or tools; customer collaboration is more important than contract negotiations; responding to changes is more important than following a plan [7]. In agile development, the risk is managed by small iterations. These help the team to adapt to unpredictable, rapidly changing requirements quickly.

Scrum

Scrum is the most well-known agile development method [7]. Scum teams usually consist of 5-10 member teams that are self-organising [8]. An iteration in a scrum is called a sprint. Sprint starts with a sprint planning meeting. Every day there might be scrum meetings to update the sprint process. A sprint review meeting is held once the sprint is finished. The scrum method helps the teams to organise more effectively, which leads to better productivity.

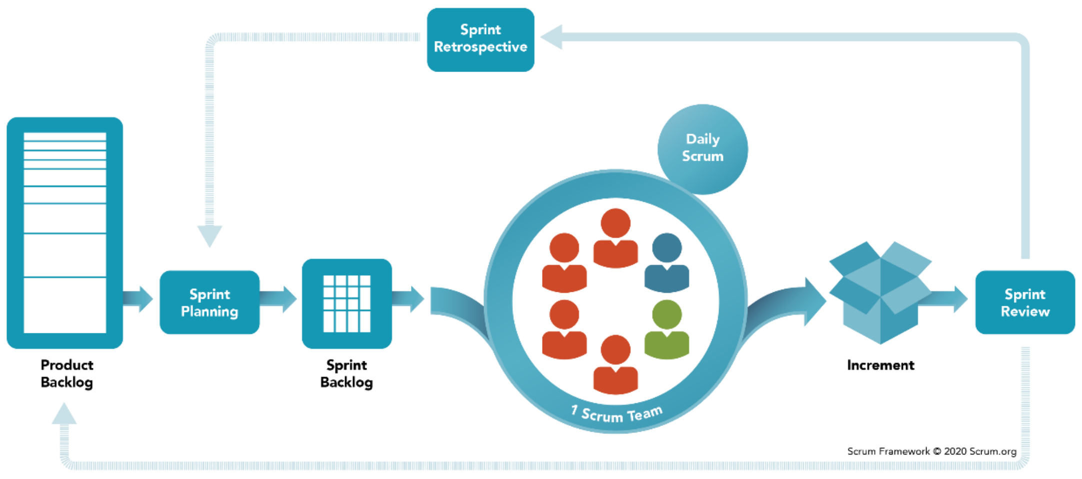


Figure 4 Scrum Framework [8]

1.1.2 Secure by design

Secure by design is an approach where security is built into the system from inception [9]. This approach starts by designing a robust security architecture, which is necessary to preserve during software evolution. Weaknesses in the software architecture can lead to various security concerns in the system—the focus shifts from finding the security bugs in the software to identifying flaws in its design. Most of the security bugs originate from weakness in the architecture.

In the 2017 article "A catalog of security architecture weaknesses", a Common Architectural Weakness Enumeration (CAWE) catalogue was proposed. Architectural weaknesses are classified into three types. Omission weaknesses are caused by a missing security tactic when necessary, e.g., storing sensitive data without encryption. Commission weaknesses refer to incorrect tactics which could result in undesirable consequences, e.g., using weak cryptography for passwords. Realisation weaknesses occur when appropriate tactics are adapted but incorrectly implemented, e.g., data is provided to the wrong session.

The CAWE catalogue was built from a list of common types of vulnerabilities. The catalogue consists of 224 architectural weaknesses, which are categorised into 11 security tactics. Designers/developers could use the CAWE catalogue as a cheat sheet to look up a security tactic adopted in their project and identify existing weaknesses within their systems.

1.1.3 Security development lifecycle

In 2004 Microsoft developed the Security Development Lifecycle (SDL) process [10]. The SDL introduces security and privacy assurance at every step of software development. It was created to reduce the number of vulnerabilities in Microsoft software. The SDL is continuously updated to take advantage of newly developed defensive techniques.

Microsoft currently has published a set of 12 practices that supply security assurance and compliance requirements. Microsoft has provided sources and tools to help developers implement their SDL process. The above mentioned 12 practices are:

1. Provide Training
2. Define Security Requirements
3. Define Metrics and Compliance Reporting
4. Perform Threat Modeling
5. Establish Design Requirements
6. Define and Use Cryptography Standards
7. Manage the Security Risk of Using Third-Party Components

8. Use Approved Tools
9. Perform Static Analysis Security Testing (SAST)
10. Perform Dynamic Analysis Security Testing (DAST)
11. Perform Penetration Testing
12. Establish a Standard Incident Response Process

1.2 Securing applications

1.2.1 Common application vulnerabilities

This thesis is focused on the security of web applications. Web application vulnerabilities differ from desktop application vulnerabilities primarily because of the connection to the internet. According to Common Vulnerabilities and Exposures (CVE), most of the web application vulnerabilities were Remote Code Execution, Denial of Service Attacks (Dos), Cross-Site Scripting (XSS), SQL Injection, File Inclusion and Cross-Site Request Forgery (CSRF) [11].

OWASP Top Ten

OWASP Top Ten [12] is an awareness document compiled by the Open Web Application Security Project (OWASP) for web application developers. It is a list of the top 10 most critical security risks to a web application. The OWASP Top Ten document was last updated in 2017.

OWASP Top 10 web application security risks are:

A1 Injection – injection flaws, such as SQL injection, occur when untrusted data is injected into a query without data validation or sanitisation.

A2 Broken authentication – authentication or session management is implemented incorrectly, allowing attackers to compromise session tokens, passwords etc., to assume users' identities.

A3 Sensitive data exposure – sensitive data, such as financial data, is not appropriately protected and can be accessed or compromised by an external party.

A4 XML external entities (XXE) – some XML processors might evaluate external references within XML documents, which can be used to disclose remote code execution, internal file shares, or denial of service attacks.

A5 Broken access control – users can access data or functionality that they are unauthorised to use.

A6 Security misconfiguration – this could be error messages containing sensitive information, default configuration, open cloud storage, etc.

A7 Cross-site scripting (XSS) – occurs when an application includes untrusted data in a web page without proper validation or escaping. It allows the attacker to hijack user sessions, deface web sites, or redirect the user to malicious sites.

A8 Insecure deserialization – can be used to perform replay attacks, injection attacks, and privilege escalation attacks.

A9 Using components with known vulnerabilities – a vulnerable component is used and given the same privileges as the application.

A10 Insufficient logging and monitoring – an attack could be unnoticed, allowing them to further attack systems, maintain persistence, and tamper with data.

OWASP Top 10 is not comprehensive enough [13]. Developers should be more aware of the issues not listed there.

CWE Top 25

The 2020 Common Weakness Enumeration (CWE) Top 25 Most Dangerous Software Weaknesses (CWE Top 25) [14] is a list of the most common and impactful issues that are reported over two previous calendar years. The list is created from Common Vulnerabilities and Exposures (CVE) data found in the National Institute of Standards and Technology (NIST) National Vulnerability Database (NVD), and the scores associated with each CVE in the Common Vulnerability Scoring System (CVSS). Each weakness is scored based on its prevalence and severity.

Below is the list of 2020 CWE Top 25 weaknesses (Table 1).

Table 1 CWE Top 25

Rank	**ID**	**Name**	**Score**
1	CWE-79	Improper Neutralisation of Input During Web Page Generation (“Cross-site Scripting”)	46.82
2	CWE-787	Out-of-bounds Write	46.17
3	CWE-20	Improper Input Validation	33.47
4	CWE-125	Out-of-bounds Read	26.50
5	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
6	CWE-89	Improper Neutralisation of Special Elements used in an SQL Command (“SQL Injection”)	20.69
7	CWE-200	Exposure of Sensitive Information to an Unauthorised Actor	19.16
8	CWE-416	Use After Free	18.87
9	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
10	CWE-78	Improper Neutralisation of Special Elements used in an OS Command (“OS Command Injection”)	16.44
11	CWE-190	Integer Overflow or Wraparound	15.81
12	CWE-22	Improper Limitation of a Pathname to a Restricted Directory (“Path Traversal”)	13.67
13	CWE-476	NULL Pointer Dereference	8.35
14	CWE-287	Improper Authentication	8.17
15	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
16	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
17	CWE-94	Improper Control of Generation of Code (“Code Injection”)	6.53
18	CWE-522	Insufficiently Protected Credentials	5.49
19	CWE-611	Improper Restriction of XML External Entity Reference	5.33
20	CWE-798	Use of Hard-coded Credentials	5.19
21	CWE-502	Deserialization of Untrusted Data	4.93
22	CWE-269	Improper Privilege Management	4.87
23	CWE-400	Uncontrolled Resource Consumption	4.14
24	CWE-306	Missing Authentication for Critical Function	3.85
25	CWE-862	Missing Authorization	3.77

The CWE Top 25 list uses only data reported publicly and captures in the NVD, and numerous vulnerabilities do not have a CVE ID. CVE/NVD does not include any vulnerabilities found and fixed before public release or internal-use software within a single organisation. These vulnerabilities might be under-represented in the list. Many CVE entries do not include enough data on the vulnerability itself, so the reported vulnerability cannot be identified with appropriate CWE. There are multiple other data biases in compiling the CWE Top 25 list.

1.2.2 Application security standards

Developers primarily focus on the functionality of the code, and security is often thought of as something that can be added later [15]. It is crucial to make developers aware of potential security vulnerabilities. A developer might not be aware of every type of vulnerability in an application and how to mitigate them. It is important to write code with security in mind. Application security standards propose a set of security requirements that help mitigate most common security vulnerabilities.

ISO/IEC 27000-series

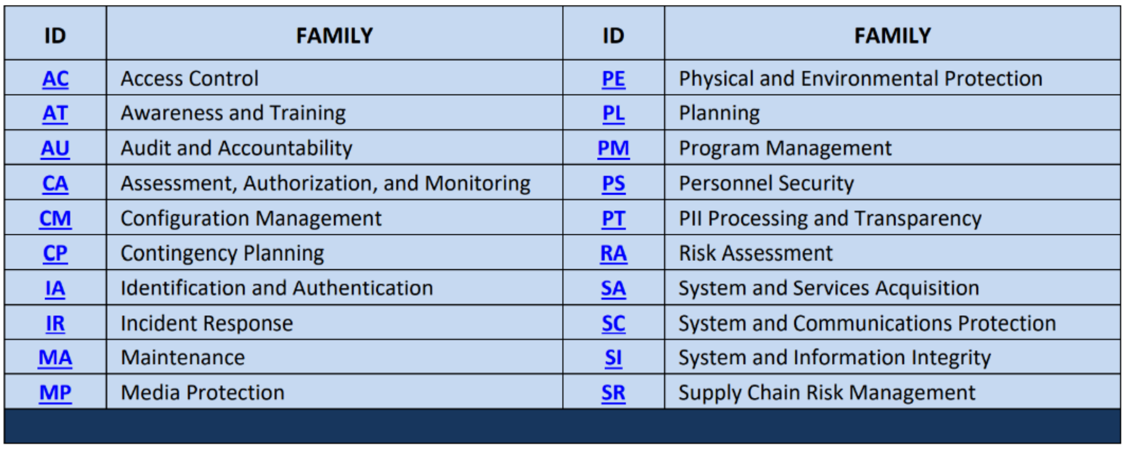
The International Organization for Standardization (ISO) is an international body composed of various standards organisations [16]. In collaboration with International Electrotechnical Commission (IEC), ISO maintains the Information Security Management system (ISMS) family of standards. These standards are used to develop and implement a framework for managing different information assets.

ISO/IEC 27001 is a standard describing ISMS requirements [17]. It introduces a set of controls in Annex A section, which are used to secure information assets. The set of controls do not provide any suggestions of how to achieve the requirements.

NIST SP 800-53

The National Institute of Standards and Technology (NIST) Special Publication (SP) 800-53 "Security and Privacy Controls for Information Systems and Organisations" establishes a list of controls that can be implemented within any organisation or system that manages information [18].

The controls in NIST 800-53 document are divided into 20 families (Figure 5). Each family contains controls specific to the topic of the family. A family consists of a set of base controls and their enhancements. The enhancements are recommendations of what to do.



ID	FAMILY	ID	FAMILY
AC	Access Control	PE	Physical and Environmental Protection
AT	Awareness and Training	PL	Planning
AU	Audit and Accountability	PM	Program Management
CA	Assessment, Authorization, and Monitoring	PS	Personnel Security
CM	Configuration Management	PT	PII Processing and Transparency
CP	Contingency Planning	RA	Risk Assessment
IA	Identification and Authentication	SA	System and Services Acquisition
IR	Incident Response	SC	System and Communications Protection
MA	Maintenance	SI	System and Information Integrity
MP	Media Protection	SR	Supply Chain Risk Management

Figure 5 Security and privacy control families [18]

NIST Cybersecurity Framework

In collaboration with industry, academia, and government, the National Institute of Standards and Technology (NIST) published *Framework for Improving Critical Infrastructure Cybersecurity*, a guide for considering cybersecurity risks [19].

The Cybersecurity Framework consists of three parts: the Framework Core, the Implementation Tiers, and the Framework Profile. The Framework Core provides a set of activities for achieving cybersecurity outcomes. It provides examples for guidance on how to achieve those outcomes. The Implementation Tiers provide context on how an organisation views the cybersecurity risks and the processes to manage the risks. The Framework Profiles are used to describe the current state of cybersecurity activities and the desired target state.

The cybersecurity activities are listed in the Appendix A section. They are divided into 5 Functions: Identify, Protect, Detect, Respond, and Recover. Every function is divided into categories that contain sub-categories that are specific cybersecurity activities.

Although the same organisation wrote NIST SP 800-53 and NIST Cybersecurity Framework, there are several differences [20]. The Cybersecurity Framework is higher level compared to NIST 800-53. Its focus is on accessing and prioritising security functions while referencing the NIST 800-53 of implementing those controls and processes. It is a great starting point for an organisation to improve its cybersecurity since it is only 55 pages than the NIST 800-53's 492 pages.

CIS Controls

The Center for Internet Security (CIS) has compiled a list of 20 high-priority, highly defensive actions that provide improvement for any enterprise seeking to improve its cybersecurity [21].

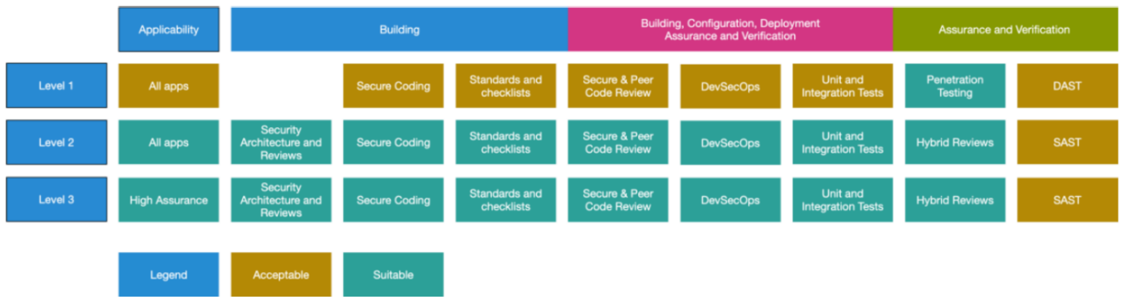
The 20 controls in the list are chosen because they stop the majority of the attacks seen today. CIS Controls follow five critical principles: offence informs defence; prioritisation; measurements and metrics; continuous diagnostics and mitigation; and automation. Every control contains many sub-controls. Implementation of a sub-control depends upon the implementation group.

There are three implementation groups. The first group is recommended for a small organisation with limited IT and cybersecurity expertise, and their data sensitivity is low. The second group is for an organisation with multiple departments or store and process sensitive data. The third group consists of organisations specialising in cybersecurity, which must address the availability, confidentiality, and integrity of sensitive data.

1.2.3 OWASP Application Security Validation Standard

The OWASP Application Security Validation Standard (ASVS) is a community-driven effort to establish a framework of security requirements and controls for developing and testing modern web applications and web services [22].

The OWASP Top 10 is a bare minimum to avoid negligence. The ASVS introduces three levels of security verification (Figure 6). Each level contains a list of security requirements mapped to security-specific controls that the developers must implement.



Applicability
Building
Building, Configuration, Deployment Assurance and Verification
Assurance and Verification
Level 1
All apps
Secure Coding
Standards and checklists
Secure & Peer Code Review
DevSecOps
Unit and Integration Tests
Penetration Testing
DAST
Level 2
All apps
Security Architecture and Reviews
Secure Coding
Standards and checklists
Secure & Peer Code Review
DevSecOps
Unit and Integration Tests
Hybrid Reviews
SAST
Level 3
High Assurance
Security Architecture and Reviews
Secure Coding
Standards and checklists
Secure & Peer Code Review
DevSecOps
Unit and Integration Tests
Hybrid Reviews
SAST
Legend
Acceptable
Suitable

Figure 6 OWASP Application Security Verification Standard 4.0 Levels [22]

The ASVS encourages the use of DAST (Dynamic Application Security Testing) and SAST (Static Application Security Testing) tools continuously throughout the development pipeline to find easy to find security issues that should never be present in an application. Business logic fails are only detected by using human assistance, meaning unit integration tests etc.

ASVS can be used in the agile development process to identify tasks that need to be implemented by the team to have a secure product. Specific ASVS requirements can be raised as a ticket/task to be visible as "debt" in the backlog that eventually needs to be done.

The Application Security Verification Standard is divided into 14 chapters, which in turn are divided into sections that contain different requirements. Level 1 only needs to comply with some of the requirements, whereas level 3 needs to comply with all the requirements.

Level 1
ASVS Level 1 is for low assurance levels. It is the only level ultimately penetration testable by humans. All other levels require access to documentation, source code, configuration, and people involved in the development process.

Level 1 is achieved by an application if it amply defends against the vulnerabilities that are included in lists like OWASP Top 10. It is a bare minimum that all applications should try to implement. Level 1 is sufficient if the application does not store or handle any sensitive data.

Level 2
ASVS Level 2 is for applications that contain sensitive data, which requires protection. It is appropriate for applications that handle business-to-business transactions, or process other sensitive assets, or industries where integrity is a critical facet of protecting their business. Most of the applications should use at least level 2 standard.

Level 3
ASVS Level 3 is reserved for applications that require significant levels of security verification, such as those in the field of military, healthcare, safety, critical infrastructure, etc. Level 3 might be needed, where failure could significantly impact an organisations operation, even its survivability.

Level 3 application requires a more in-depth analysis of its architecture, coding, and testing. It should be modularised in a meaningful way, and the security of each module should be adequately documented. Controls for ensuring confidentiality, integrity, availability, etc., should be implemented.

1.3 Attacks on software supply chain

Software is always in a constant state of work in progress. It relies on patches and updates addressing bugs or vulnerabilities and making functional improvements. Constant improvements require a supply chain that might be vulnerable to attacks.

Since 2019, the Atlantic Council's Cyber Statecraft Initiative has maintained the Breaking Trust project [1]. This project catalogues software supply-chain intrusions over the past decade to identify significant trends in their execution. Broken Trust project has determined that software supply-chain exploitations have become more frequent and more impactful over the decade and their targets have become even more diverse [23].

1.3.1 PHP git repository

On the 28th of March in 2021, the official PHP git repository was compromised [24]. Two malicious commits were pushed to the *php-src* repository[1] under the accounts of Rasmus Lerdorf, creator of PHP scripting language [25], and Nikita Popov [24]. It is speculated that instead of compromised git accounts, the git.php.net server was compromised. Following the hack, it was decided that the use of the *git.php.net* server will be discontinued, and the GitHub repository will become canonical. Now every contributor will need to be a member of the PHP organisation on GitHub.

1.3.2 CCleaner

In August of 2017, hackers managed to embed malicious code into CCleaner before it was compiled and released to the users [26]. The compromised installers had malware attached to them designed to collect information from the computer and install a secondary payload on systems.

[1] https://github.com/php/php-src/commit/c730aa26bd52829a49f2ad284b181b7e82a68d7d and https://github.com/php/php-src/commit/2b0f239b211c7544ebc7a4cd2c977a5b7a11ed8a

At first, it was thought that the goal was to infect as many users as possible. More than 2 million copies of the infected installer were used. Later investigation revealed that the secondary payload was installed only on 40 targets, suggesting that this attack was targeted at a specific group of users [27]. The malware contained a list of domains that would have received the second-stage payload. Those included were Samsung, Microsoft, Sony, and others, indicating that the goal of this attack was espionage.

The hackers initially got access to the company's network using stolen credentials to log into a TeamViewer remote desktop account on a developer's PC [28]. Attackers only worked outside of office hours when it was unlikely that people would be using the targeted machines. They installed malware on the computers with keylogger functionality and, after some months, began to contaminate CCleaner downloads. It is believed that a cyberespionage group known as APT17 was behind this attack [29]. It is believed that China's intelligence coordinates this group.

1.3.3 Kingslayer

In 2017 RSA Security researchers disclosed that an administrative software package called EvLog was involved in a software supply-chain intrusion [1]. EvLog is a software mainly used by system and domain administrators, which made it a valuable target. It is believed that targets were high-tier clients, but it is still unknown who might have been affected.

During the attack in 2015, the update downloads were subverted to a malicious version of the software [30]. This attack was realised via an *.htaccess* redirect that pointed to a website controlled by the malicious actors, where signed versions of the application executable containing the Trojan were hosted.

The malware allowed for secondary payloads to be loaded onto the compromised systems. Since the affected application was used and trusted by many system administrators, it is still unknown how many systems might have been affected by this hack.

1.3.4 Flame

Flame malware, found in 2012, exploited the use of the MD5 hash function to forge a valid certificate for the Microsoft Windows Update service [31]. MD5 hash function's weakness has been known for 20 years, but it still used by systems today. This vulnerability allowed a backdoor into Windows. The flaw was in Microsoft's Terminal Services licensing certificate authority (CA) that allowed them to generate code-validating certificates allegedly signed by Microsoft [32].

The Flame malware was found infecting systems in Iran and other countries in the Middle East and North Africa [33]. It is believed that the malware was in use as a cyberweapon to disrupt Iran's nuclear program.

Flame malware recorded every possible conversation had through the computer and stored frequent screenshots of activity on the machine. The malware also scanned the local network to collect usernames and passwords. The final stage destroyed the system whipping it completely clean with no traces of malware on it.

The Flame malware is believed to be written in partnership between Israel and the United States to sabotage Iran's uranium enrichment program.

1.3.5 Able Desktop

In 2020 it was disclosed that Able Software's Able Desktop application was compromised using hijacked updates [1]. This hack affected most of the Mongolian government, including the Office of the president and Ministry of Justice. Able Desktop application updates were unsigned, and for this reason, intruders did not need to steal or forge an update signature. It is believed that an update server of the Able Desktop application was compromised, and the legitimate update replaced by malware [34].

1.3.6 WIZVERA VeraPort

WIZVERA VeraPort is an application for a security plugin for verifying the identity of the users, required to be used by South Korean banking and government websites [1]. Users are blocked from using these sites unless they have the application installed on their devices.

The attackers compromised a website that is required to use WIZVERA VeraPort [35]. Once the web server is compromised, the visiting user gets the malicious binary. The attackers used a valid code-signing certificate that was likely obtained using spear-phishing attacks to sign malware samples sent to affected users.

1.3.7 Juniper

In 2015 a severe flaw was discovered in the Juniper Network NetScreen line of products [1]. It was discovered that hackers infiltrated Juniper's software supply chain and compromised an algorithm used to encrypt classified communications. NetScreen manufactures security systems provide firewall service, virtual private network (VPN) service, and network management to large organisations.

In 2006 NIST released an encryption algorithm developed by National Security Agency (NSA), which relies on a static value "Q" to encrypt data. Juniper used only this algorithm to ensure the security of their products. In 2012 malicious actors infiltrated Juniper's development process and changed the "Q" value to the one they knew, which allowed them to decrypt all traffic using Juniper VPNs.

At the time of the attack, clients for this product included multiple important US agencies, like the Federal Bureau of Investigation (FBI). The exploit went unnoticed for three years. NSA was aware of this "Q" value backdoor in their encryption algorithm and might have intentionally left it there for its intelligence purposes [36].

1.3.8 Sunburst

The most recent of these software supply-chain attacks targeting big companies is the infiltration of the SolarWinds' Orion product deployment system found in December of 2020 [1]. The campaign, named Sunburst, is still ongoing while writing this thesis. Sunburst uses Microsoft Identity and Access Management (IAM) products to move through organisations and their inboxes.

Sometime in 2019, malicious actors gained access to SolarWinds' software development and build infrastructure and inserted a back door into a version of Orion software via a change to a dynamic-link library (DLL). SolarWinds distributed a digitally signed compromised version of Orion for three months. This infection affected systemically important vendors, like Microsoft and Intel, and multiple US federal government agencies, like Homeland Security.

1.4 Existing research

In 2020 a master thesis was submitted to the University of Oslo that assessed the security of an open-source health management platform using the OWASP Application Security Verification Standard (ASVS) [37]. The research was conducted by interviewing the developers of this project. The developers were asked to assess the ASVS controls and whether these controls apply to the software. All the applicable controls were then given a flag of a pass or fail. Each of the failed control was then evaluated and ranked by severity. The thesis recommends further research into implementing security into the Software Development Lifecycle (SDLC), which this thesis aims to do.

2 Research method

This research aims to demonstrate if using an application security verification standard is sufficient to secure against most frequent software supply-chain attacks. In this chapter, the SimplBooks web-based accounting software is examined as a case study.

In the first part of this chapter, the chosen study method is discussed. Second, the overview of the chosen study subject is given by examining the company's history, the setup of the software, the currently used development process and possible shortcomings of the process.

2.1 Methodology

This thesis is conducted using the applied descriptive study method [38]. The applied study aims to develop new procedures, techniques or products.

The data collection method for this research is descriptive research. The use of this research method means that data is gathered without controlling any of the variables. An applied descriptive study observes how the application of knowledge, process, or system work in a real-life setting. Because of the real-life setting, the results have a higher degree of realism than controlled experimentation.

A case study is a type of applied descriptive study. An applied case study focuses on the expected outcome of a specific event. A case study can be used as a step-by-step guide to achieving the desired results, but the results may vary as case studies are flexible in design.

In chapter 1, a literature review was conducted to provide an overview of current knowledge and identify gaps in existing research. In the literature review, multiple application security standards were analysed for simplifying the selection of a standard for application in this thesis. As the aim of this thesis is to secure the development lifecycle, multiple cases of software supply-chain attacks were analysed.

For the data collection procedure, an audit is conducted to identify the current state of the application and its security. The audit will provide the necessary data for the analysis procedure. The standard integration will contain analysis of what steps should be taken so the development process will be in compliance with the selected standard.

2.2 Research subject

The case study is conducted on an accounting software based in Estonia called SimplBooks. The author of this thesis has been working in the company as a developer for almost five years. As a long time developer, they can make changes to the development process and conduct an audit of how the SimplBooks application meets the requirements of the OWASP ASVS.

The SimplBooks software structure is discussed in appendix 1 (Appendix 1 – SimplBooks software structure). The main framework and the libraries used by the software is discussed in appendix 2 (Appendix 2 – Framework and libraries used by SimplBooks software).

2.2.1 Company overview

SimplBooks OÜ was founded in 2012[1] by Jaanus Reismaa and Rene Meres. It was preceded by eAktiva, founded in 2009[2], the first online Estonian software for accounting and invoicing. eAktiva was later renamed and updated to become SimplBooks[3]. In 2017 its subsidiary, SimplBooks OY, was founded in Finland[4].

SimplBooks [39] is designed to be used by micro and small companies and accounting bureaus (Figure 7). SimplBooks offers low-level client management, invoicing, automated accounting, warehouse management, wages, and many other features. SimplBooks currently has over 10 000 users making it one of the largest online accounting software in Estonia. SimplBooks also collaborates with multiple universities

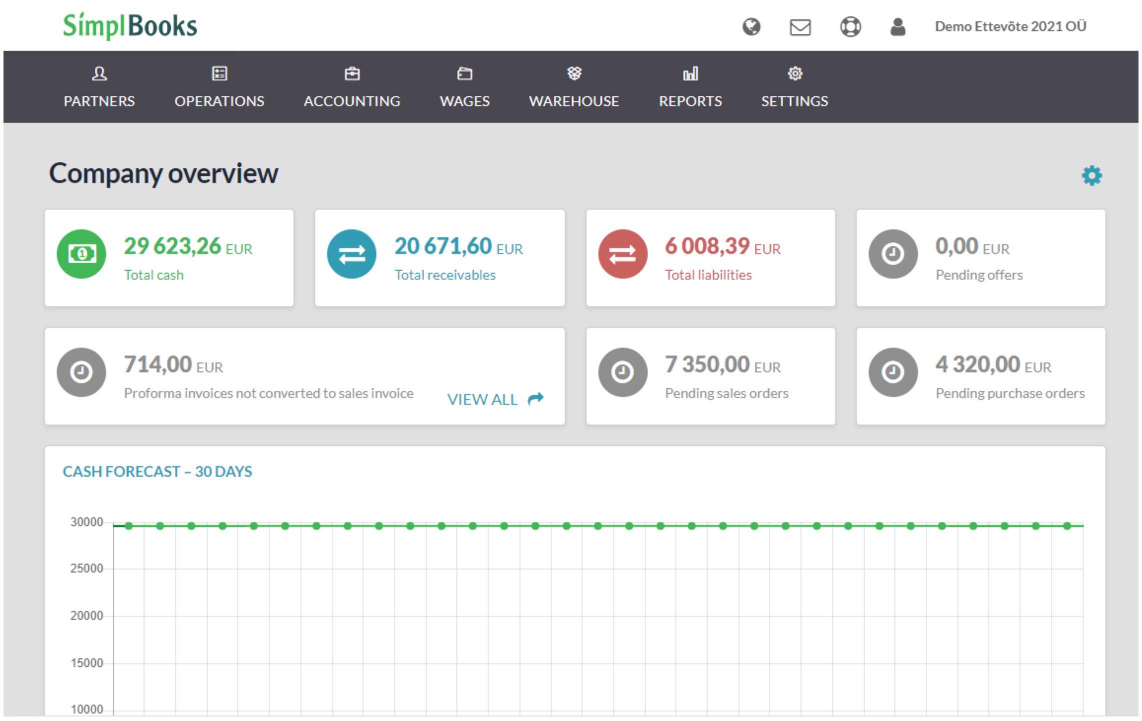
[1] https://ariregister.rik.ee/ettevotja?id=3000040897

[2] https://twitter.com/SimplBooks/status/5172771795

[3] https://twitter.com/SimplBooks/status/80882622565318656

[4] https://www.finder.fi/IT-konsultointi+IT-palvelut/SimplBooks+Oy/Helsinki/yhteystiedot/3163636

in Estonia and Finland, including Tallinn University of Technology and Estonian Business School.



Company overview

29 623,26 EUR
Total cash

20 671,60 EUR
Total receivables

6 008,39 EUR
Total liabilities

0,00 EUR
Pending offers

714,00 EUR
Proforma invoices not converted to sales invoice

7 350,00 EUR
Pending sales orders

4 320,00 EUR
Pending purchase orders

CASH FORECAST – 30 DAYS

Figure 7 SimplBooks application

2.2.2 Development lifecycle

SimplBooks uses an agile development process inspired by the Scrum methodology. All the code is stored in private GitHub repositories. Tasks are categorised into six stages: *waiting*, *to-do*, *in-development*, *code review*, *testing* and *done*. The development lifecycle is described in the figure below (Figure 8).

All the tasks on the *waiting* list are given a score using the ICE Scoring Model [40] for better prioritisation. ICE stands for impact, confidence, and ease. Each of these is given a numerical value on a scale of 1-10 (low to high). Impact means that how many customers are impacted or will be impacted by the bug/feature. Confidence is the grade for certainty that the task will have the predicted impact. Ease is the level of effort to complete the project. Customer support is expected to score impact and confidence for all tasks because they know what the clients expect from the application and their experiences. The development team scores ease by estimating the time it takes for the task to be completed, from analysing to writing code to testing. The time estimated is then calculated into a score from 1-10. Once all the scores are given, the overall score for the task is calculated using the formula $score = I * C * E$. The tasks are then reordered based on the score from highest to lowest. Significant feature updates are excluded from this and are added onto *to-do* based on a previously constructed roadmap.

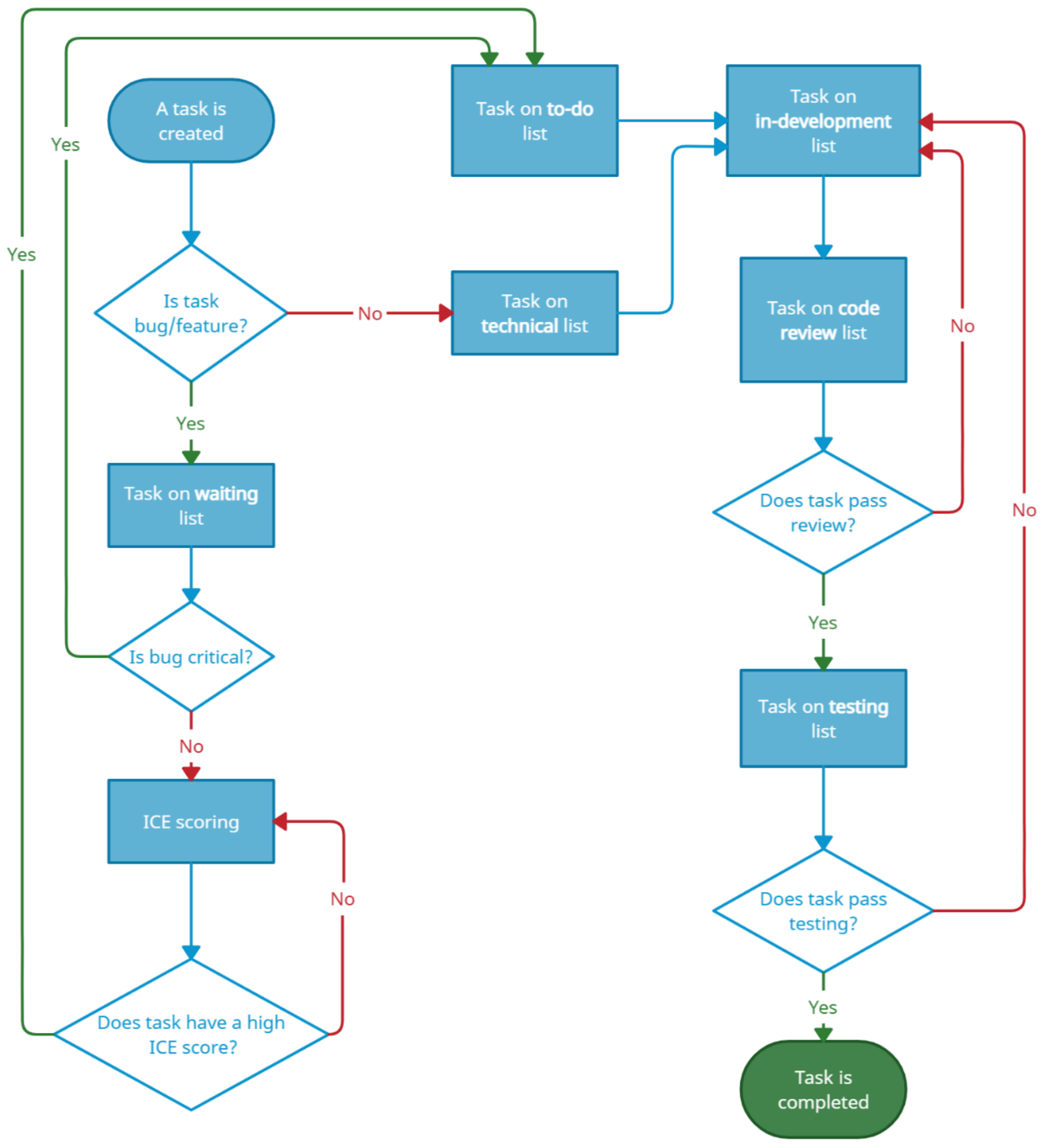


Figure 8 SimplBooks development lifecycle

Code changes that are only technical and do not change any functionality of the code, e.g., updating a version of a library used, will never be listed in the *waiting* list. There is a separate list of technical tasks from where a developer should try to work on at least one task per week for these changes.

Every Monday, the development team goes over the *in-development* and *to-do* lists to determine how many tasks to add to the *to-do* list from the *waiting* list. The *to-do* list contains the tasks that are expected to be worked on in the current week. Urgent bugs are added to the *to-to* list skipping the *waiting* list. The analyst analyses the feature tasks before they are eligible for the *to-do* list. The tasks under the *to-do* list are usually left unassigned, meaning that no particular developer is assigned to work on a given task.

The tasks that the developers are working on are listed in the *in-development* list. The tasks in this list are assigned to the developer who is currently working on them. Developers use Vagrant[1] virtual development environments for testing the application and PhpStorm[2] for writing code. Developer creates a new git branch onto the master branch where they commit all the changes related to the task. If the task s a feature update, then the developer must also write unit tests to test the output of new functions. Once changes are committed to the branch, a pull request is created, and the ask is moved to *code review*.

The *code review* list contains the pull requests waiting for review from other developers. The task in review is assigned to the developer, who should make the code review. When reviewing the code, there is a checklist that should be followed. The reviewer should confirm that new code written follows the following checklist: Clean Code principles are used, the database requests are optimised, *FIXME* and *TODO* comments are resolved and removed, debugging outputs such as *console.log()* are removed, error handling is done correctly, CSS classes replace HTML style attributes, JavaScript uses as few global variables as possible, etc. Once the code review is done, the pull request is approved, or some changes are requested from the developer. Once the task has passed the code review, it moves on to the *testing* list.

The *testing* list contains all the pull requests that are waiting for testing or are being tested currently. The tester assigns the task they are working on to themselves. Tests are conducted on a development server separate from the live server. The tester tests the business logic and the expected outcome of the task. If the task includes making changes

[1] https://www.vagrantup.com/

[2] https://www.jetbrains.com/phpstorm/

in the user interface (UI), then the tester should check if the new views work on all screen sizes and match the UI of the rest of the application. The tester should check if the input validation works as expected by entering incorrect data into the form inputs. In case of significant feature updates, the tester should add automated browser tests using Ghost Inspector[1]. The tester should keep an eye on the environment error logs to catch any errors, notices, or warnings hidden from the end-user. Once the testing has concluded, the tester is expected to write a testing report. The report should contain what was tested and what were the results. If there are errors found in the testing phase, the task is assigned to the original developer to correct the mistakes. If the testing is successful, then the task is assigned to the original developer, who then merges the pull request into the master branch and moves the task to the *done* list.

The *done* list contains all the tasks that are ready to be deployed within the next release. Once a week, a deployment is conducted to update the application and its components. The deployment has its own checklist. A developer is assigned to conduct the code review of the whole code before deploying. The developer should check that every task in the *done* list has been tested and merged into the master branch, database schema migrations are good, database data migrations are good and turned off for new environments, default data SQL is updated with new data, new database columns have indexes, all the unit tests pass, GitHub Actions workflow tests have passed without errors. After this, new translation files are generated, and new phrases are translated. After merging new translation files, a GitHub release is created from the master branch. The release is uploaded to the administration application. As a final test, a new environment is created in the live server to see if there are any problems. Once this test is completed, the update is uploaded into every environment. A changelog is written based on the task list in the *done* section and is uploaded into the internal Wiki. Afterwards, all the tasks in the *done* list are marked "Completed".

This development lifecycle is undoubtedly not entirely secure. The concerns for this development lifecycle's security are further discussed in appendix 3 (Appendix 3 – Concerns in the SimplBooks development lifecycle).

[1] https://ghostinspector.com/

2.3 Standard selection

In the 2012 paper “Reducing Attack Surface of a Web Application by Open Web Application Security Project Compliance”, the authors proved that the attack surface of a web application could be reduced by 41% by applying the OWASP Top 10 [41]. In an article by Ferda Özdemir Sönmeza, the OWASP ASVS version 3 compliance check was used as the alternative to the attack surface calculations to determine the vulnerability level [42].

SimplBooks is a web-based accounting application; therefore, OWASP Application Security Verification Standard (ASVS) is applicable here since it aims to secure web applications [22]. Currently, SimplBooks does not follow any standards in its development process. SimplBooks needed a standard that applies to the whole development chain and not explicitly aimed at a particular goal. The aims of CIS controls and NIST Cybersecurity Frameworks is to protect against cyberattacks. As a relatively small team develops the SimplBooks application, it was also crucial that the security verification standard is simple to use and understand. The ISO 27000-series standards are very long and complicated for a simple developer to use.

As the ASVS is selected, it is essential to determine which level of ASVS will be applicable for the SimplBooks application. Level 1 of the standard is considered to be the bare minimum requirements for any web application. Level 2 standard is aimed at applications that handle significant business-to-business transactions, implement business-critical functions or process other sensitive assets where integrity is a critical facet to protect. Based on this, level 2 should be the standard used in SimplBooks as the data processed by the application is business-critical for the users, and SimplBooks handles integration with banks and e-invoice providers.

2.4 Software audit

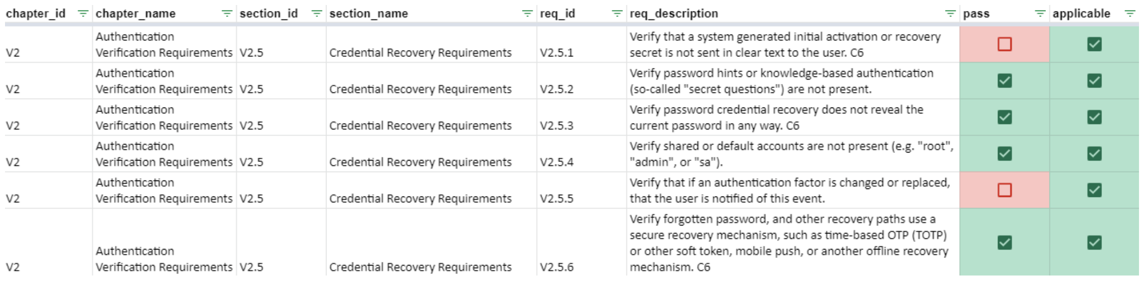
The audit is essential to map out which of the ASVS requirements apply to the application and which requirements are already met. For the simplification of the audit process, the level 1 requirements are audited first. The level 2 requirements will be audited last. The audit was conducted using a Google Sheets document.

2.4.1 Level 1 audit results

As Level 1 controls are the bare minimum of web application security, it is vital to map out what requirements are already met. Level 1 contains 131 requirements in total. There are no requirements for chapters V1 *Architecture, Design and Threat Modeling Requirements*, and V6 *Stored Cryptography Verification Requirements*.

The first step of the audit was to control if the requirements were applicable for the SimplBooks software. Of 131 requirements in Level 1, 120 were determined to be applicable. Chapter V2 *Authentication Verification Requirements*, section V2.7 *Out of Band Verifier Requirements* was determined not to be applicable because the application does not use two-factor authentication. Other requirements were also deemed not applicable because the functionality described is not used by the application, e.g. V13.3 *SOAP Web Service Verification Requirements*.

The second step was to determine if the requirement is met or not. The framework documentation was read, the SimplBooks application code was checked, and the application manually tested. Of the 120 applicable requirements, 86 or 71.67% were met. For most chapters, over half of the requirements were met except for chapters V3 *Session Management Verification Requirements* and V11 *Business Logic Verification Requirements,* which did not contain many requirements. Below is an example of the section V2.5 *Credential Recovery Requirements* audit (Figure 9). From this table, it can be seen that there is no email sent to a user in case of password change.



chapter_id	chapter_name	section_id	section_name	req_id	req_description	pass	applicable
V2	Authentication Verification Requirements	V2.5	Credential Recovery Requirements	V2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user. C6	☐	☑
V2	Authentication Verification Requirements	V2.5	Credential Recovery Requirements	V2.5.2	Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.	☑	☑
V2	Authentication Verification Requirements	V2.5	Credential Recovery Requirements	V2.5.3	Verify password credential recovery does not reveal the current password in any way. C6	☑	☑
V2	Authentication Verification Requirements	V2.5	Credential Recovery Requirements	V2.5.4	Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").	☑	☑
V2	Authentication Verification Requirements	V2.5	Credential Recovery Requirements	V2.5.5	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.	☐	☑
V2	Authentication Verification Requirements	V2.5	Credential Recovery Requirements	V2.5.6	Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as time-based OTP (TOTP) or other soft token, mobile push, or another offline recovery mechanism. C6	☑	☑

Figure 9 V2.5 *Credential Recovery Requirements* audit

The overall results of the audit have shown there are multiple facets of the application where security can be improved. The audit also shows that most of the requirements are met. Below is the summary table for the Level 1 requirements (Table 2). With red are marked the chapters of the standard where less than half of the requirements were met, yellow marks chapters where half or more of the requirements are met, green marks the chapters where all the requirements are met.

Table 2 Level 1 audit

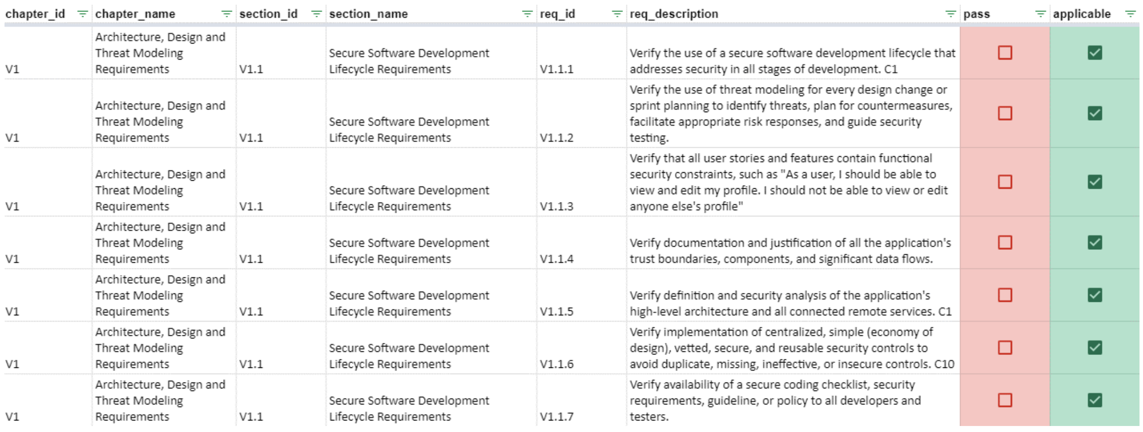
Chapter	**Passed**	**Failed**	**Applicable**	**All**
V1	0	0	0	0
V2	13	8	21	27
V3	5	6	11	12
V4	7	2	9	9
V5	22	3	25	27
V6	0	0	0	1
V7	2	1	3	3
V8	7	0	7	7
V9	3	0	3	3
V10	2	1	3	3
V11	3	2	5	5
V12	8	3	11	11
V13	6	0	6	7
V14	8	8	16	16
Total	**86**	**34**	**120**	131

2.4.2 Level 2 audit results

Level 2 contains 267 requirements, of which 136 were not in Level 1. Most of these requirements are under chapters V1 *Architecture, Design and Threat Modeling Requirements* (41 requirements) and V2 *Authentication Verification Requirements* (26 requirements).

The Level 2 audit was conducted the same as the Level 1 audit, except that the organisation's internal documents were also reviewed. Out of the 136 Level 2 requirements, 90 were evaluated to apply to the application. From chapter V2 *Authentication Verification Requirements*, out of the 26 requirements, only nine were considered to be applicable. SimplBooks application does not use Look-Up Secret verification or one-time passwords (OTPs).

Of the 90 requirements determined to be applicable, 61 or 67.78% of the requirements were met. Most of the failed requirements are from the chapter V1 *Architecture, Design and Threat Modeling Requirements*. From section V1.1 *Secure Software Development Lifecycle Requirements*, it can be seen why adopting this security standard is essential because none of the requirements is met (Figure 10).



chapter_id	chapter_name	section_id	section_name	req_id	req_description	pass	applicable
V1	Architecture, Design and Threat Modeling Requirements	V1.1	Secure Software Development Lifecycle Requirements	V1.1.1	Verify the use of a secure software development lifecycle that addresses security in all stages of development. C1	☐	☑
V1	Architecture, Design and Threat Modeling Requirements	V1.1	Secure Software Development Lifecycle Requirements	V1.1.2	Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.	☐	☑
V1	Architecture, Design and Threat Modeling Requirements	V1.1	Secure Software Development Lifecycle Requirements	V1.1.3	Verify that all user stories and features contain functional security constraints, such as "As a user, I should be able to view and edit my profile. I should not be able to view or edit anyone else's profile"	☐	☑
V1	Architecture, Design and Threat Modeling Requirements	V1.1	Secure Software Development Lifecycle Requirements	V1.1.4	Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.	☐	☑
V1	Architecture, Design and Threat Modeling Requirements	V1.1	Secure Software Development Lifecycle Requirements	V1.1.5	Verify definition and security analysis of the application's high-level architecture and all connected remote services. C1	☐	☑
V1	Architecture, Design and Threat Modeling Requirements	V1.1	Secure Software Development Lifecycle Requirements	V1.1.6	Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls. C10	☐	☑
V1	Architecture, Design and Threat Modeling Requirements	V1.1	Secure Software Development Lifecycle Requirements	V1.1.7	Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.	☐	☑

Figure 10 V1.1 Secure Software Development Lifecycle Requirements audit

From the Level 2 audit, it can be seen that most of the requirements that failed are documentation requirements. Below is the summary of Level 2 requirements (Table 3).

Table 3 Level 2 audit

Chapter	Passed	Failed	Applicable	All
V1	19	14	33	41
V2	7	2	9	26
V3	1	2	3	6
V4	0	0	0	1
V5	3	0	3	3
V6	4	1	5	12
V7	7	3	10	10
V8	2	2	4	8
V9	4	0	4	4
V10	2	0	2	2
V11	1	2	3	3
V12	2	0	2	4
V13	4	1	5	8
V14	5	2	7	8
Total	**61**	**29**	**90**	136

2.5 Standard integration

From the audit, it can be seen that 63 requirements need to be implemented into the SimplBooks development lifecycle. Integrating these requirements will be a gradual slow process that will take time to be fully completed.

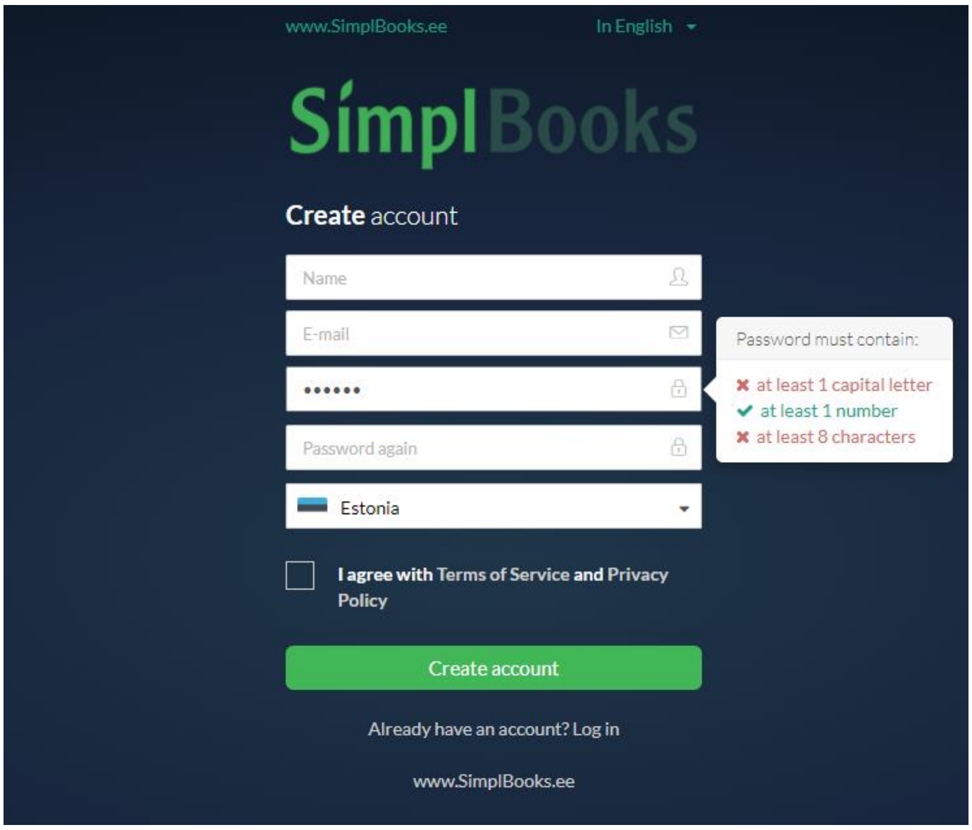
A review of all the failed requirements for Level 1 was conducted. The failed requirements were categorised into three groups: technical, feature, and documentation. Technical requirement tasks mean changes to the application that do not change the user interface (UI) and user experience (UX). Feature requirement tasks require changes to be made in UI and UX. Documentation requirement tasks require creating a document or checklist that the software development team should follow.

2.5.1 Level 1 requirements

Level 1 requirements are more technical than Level 2. Technical requirement changes do not need to go into a waiting list to be scored using the ICE model. These tasks are in a separate list where the developer should work on at least one task per week. For Level 1, 25 of the 34 requirements were identified to require technical changes to the application.

Nine of the unimplemented requirements are identified as a feature change. The requirements that require changing the behaviour of the application cannot be implemented immediately. The tasks for these requirements must first move to the waiting list to be scored using the ICE model. Once the tasks have been scored, if needed, they will be analysed by the analyst. After that, the tasks will be ready for development as per the previously described development lifecycle process.

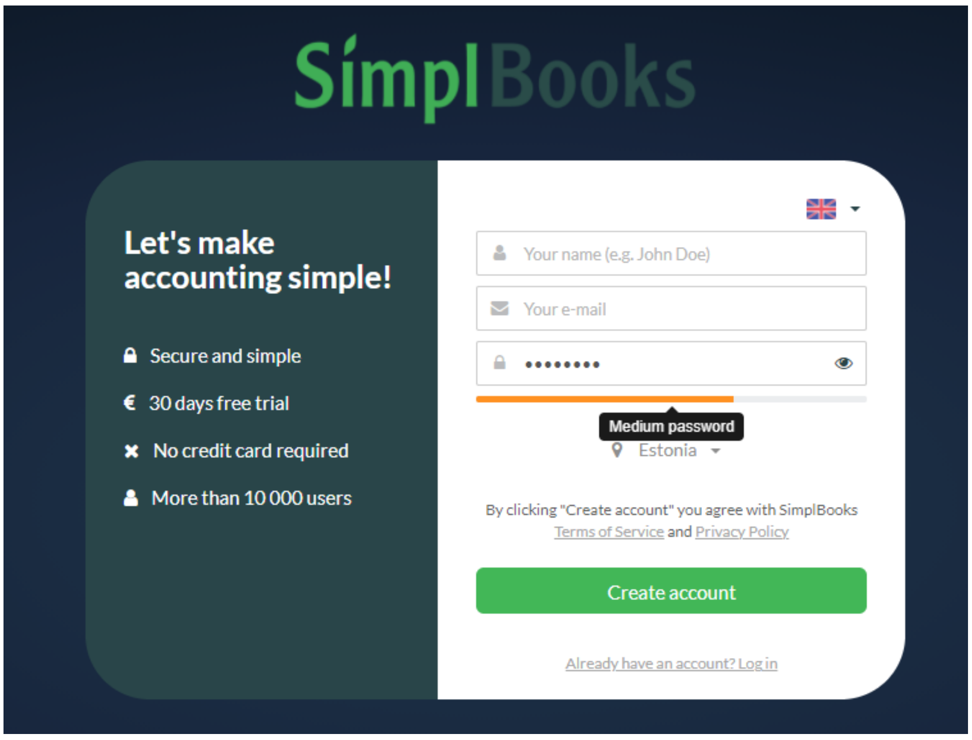
From the V2.1 *Password Security Requirements* section, most of the identified feature requirements are already in development in part of a registration form update. Currently existing form fails the requirements V2.1.8, V2.1.9 and V2.1.12 (Figure 11). The new registration form that is in development will remove the existing password composition rules, provide a password strength meter and allow the user to unmask their entered password (Figure 12).



Password must contain:

✖ at least 1 capital letter
✓ at least 1 number
✖ at least 8 characters

Figure 11 SimplBooks registration form

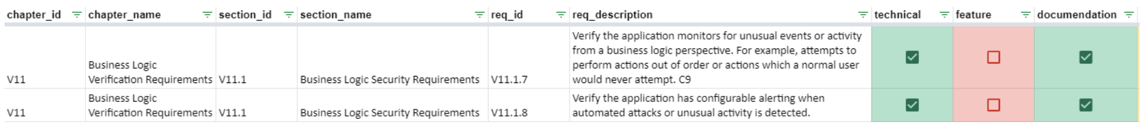


By clicking "Create account" you agree with SimplBooks Terms of Service and Privacy Policy

Figure 12 SimplBooks new registration form

2.5.2 Level 2 requirements

Level 2 requirements require documentation changes as well as technical and feature changes. Of the 29 Level 2 requirements not implemented, 11 require documentation update. These tasks will not go through the typical development lifecycle and will be worked on separately. Some of these documentation updates are technical changes like V11.1.7 and V11.1.8, requiring a threat model or abuse case definition (Figure 13).



chapter_id	chapter_name	section_id	section_name	req_id	req_description	technical	feature	documendation
V11	Business Logic Verification Requirements	V11.1	Business Logic Security Requirements	V11.1.7	Verify the application monitors for unusual events or activity from a business logic perspective. For example, attempts to perform actions out of order or actions which a normal user would never attempt. C9	☑	☐	☑
V11	Business Logic Verification Requirements	V11.1	Business Logic Security Requirements	V11.1.8	Verify the application has configurable alerting when automated attacks or unusual activity is detected.	☑	☐	☑

Figure 13 V11.1 Business Logic Security Requirements classification

Section V1.1 *Secure Software Development Lifecycle Requirements* is the main section in Level 2 that needs attention. In this section, the use of a secure software development lifecycle is one of the requirements that can be only met if the application passes the ASVS audit. OWASP Top 10 Proactive Controls recommends implementing the security requirements iteratively over time [43]. Use of threat modelling and use stories is required and should be done by the analyst when planning a new feature. The requirement V1.1.7 sets that a checklist or a guideline is introduced to all the developers and testers. As of now, SimplBooks has a checklist for code review and testing.

All the requirements needing technical changes will be listed in the technical tasks list but will have less importance than the Level 1 tasks as those are critical for the base security level. The requirements that need feature changes, such as UI or UX, will be added to the waiting list for ICE scoring and analysis. The order of these tasks will depend on the ICE score given.

3 Results

As previously discussed in chapter 1, software supply chain attacks have become more prevalent in the last decade. The goal of this thesis was to determine if the usage of an application security standard is, in fact, enough to defend against the most popular types of software supply chain attacks. Before it is possible to determine if the application security standard gives enough protection, it is vital to determine which types of attacks are most common.

Broken Trust report by Atlantic Council gives an overview of the software supply chain attacks that have been identified since 2010 (Figure 14)[1]. From their report, the most common software supply-chain attacks can be identified. Most of the attacks are conducted by hijacked updates. It is also apparent that open source repositories could be a vulnerability. Thirty-six of the vulnerabilities have been introduced in system design and implementation progress.

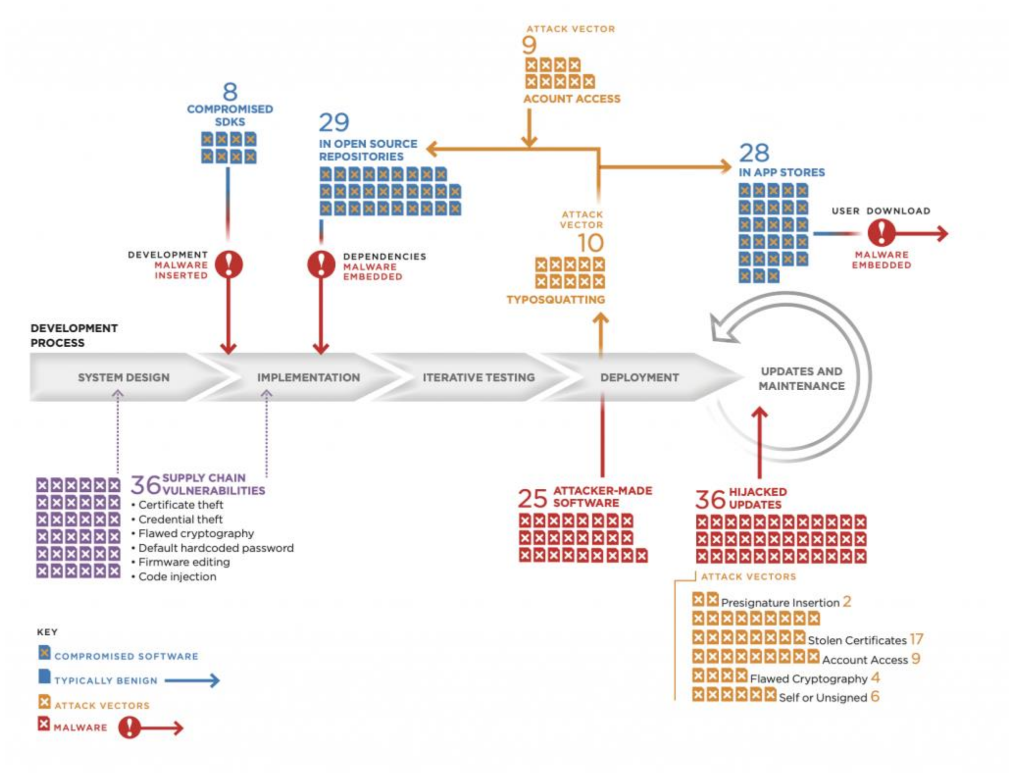


Figure 14 Dataset of software supply-chain attacks and disclosures

Open source projects are a popular choice for selecting a library to be used in a project. The attackers might use two different strategies to inject a package into a dependency tree [44]. They might infect an existing package or submit a new package.

Latter of these can be achieved by using typosquatting. Typosquatting is an attack where a name of a popular package changes a little, e.g., deleting *-o* from *crypto*, and used as a name for an infected package [45]. ASVS requirement 14.2.4 stated that third-party components must come from pre-defined, trusted and continually maintained repositories. ASVS requirement 14.2.5 requires that an inventory catalogue is maintained of all the third-party libraries in use. These two requirements might help to mitigate a typosquatting attack.

Open source projects rely on a community of contributors who propose changes to the project's codebase. These changes are then reviewed and merged into the project's codebase. Attackers might mimic being a contributor by submitting a pull request (PR) with a bug fix or seemingly helpful feature/dependency to inject malicious code [44]. This PR might be approved and merged into the projects main codebase. An attacker could also use weak or compromised credentials to commit malicious code into the project's codebase. This attack was used in compromising the official PHP git server, which was previously mentioned in chapter 1.3. In all of these cases, the malicious code will become a part of the following official package release and, therefore, might infect all the projects using that release. There are no ASVS requirements that could mitigate this type of attack. Instead, it is up to the project maintainer to maintain the security of their project.

The most vulnerable step in the development process is the update and maintenance process [1]. In chapter 1, multiple different compromised systems were discussed where the hijacking of the update process was the culprit. In the case of the Able Desktop application, the update server of the application was breached, and the legitimate update was replaced by malware. In the case of a web application, no update server could be compromised since the updates for web applications are handled by the application maintainer. For web applications, the build and development processes might be compromised. In ASVS section V14.1 *Build* requires the security verifications of mentioned processes.

Stolen certificates and stolen credentials could give access to the implementation step of the development process. In the case of CCleaner, the attacker got hold of stolen certificates and managed to inject malware into the application before the build process [1]. The attack was conducted using TeamViewer on a developer's computer outside office hours. ASVS requirement 1.1.1 requires the use of a secure development lifecycle that addresses security in all stages of development.

Flawed cryptography could be an access point for attackers, as demonstrated in Flame and Juniper attacks. In the case of the Flame malware, the usage of a weak hash function MD5 was taken advantage of [1]. In Juniper's case, they trusted an algorithm made by the NSA without using any other methods for encryption. It came out that NSA had designed a backdoor into the algorithm that the attackers then exploited. The usage of algorithms is set by the ASVS section V6.2 *Algorithms*. It gives different recommendations of which algorithms not to use.

The OWASP Application Security Verification Standard is good enough to secure against possible supply-chain attacks. For most of the most frequently detected software supply-chain attacks, specific requirements in the ASVS document can be found. The OWASP ASVS does not have requirements specific to the organisation's operations, e.g., training the personnel on cybersecurity. This means that phishing attacks might be a possible vulnerability to the system.

4 Discussion

In the process of conducting the case study, there were multiple observations made of the ASVS, its applicability and coverage of the supply-chain attacks, and the development process overall. In this chapter, these observations will be discussed, and some propositions will be made.

4.1 Shortcomings of the ASVS

The ASVS document contains many requirements, but some of the requirements are too broad to be verifiable. The very first requirement, 1.1.1, says to "Verify the use of a secure software development lifecycle that addresses security in all stages of development." There is no explanation of what should be considered as a "secure software development lifecycle". The ASVS document recommends using OWASP Proactive Controls first control *Define Security Requirements* as the basis, but that document, in turn, recommends using ASVS. There should be a separate chapter about the requirements for a secure software development lifecycle, with each development stage having its section of requirements.

The current version of the document, version 4.0.2, is still incomplete as of May 2021. It contains placeholders for planned requirements such as V1.3 *Session Management Architectural Requirements* or V1.12 *API Architectural Requirements*. The ASVS used to contain Internet of Things (IoT) verification requirements, but this section has since been moved to the appendix, and the OWASP IoT project is recommended instead.

The OWASP ASVS is a framework mainly compiled of community ideas and feedback. The OWASP organisation maintains a GitHub project of the ASVS document[1]. The ASVS GitHub project is open for everyone to give feedback or share their ideas. It is

[1] https://github.com/OWASP/ASVS

recommended to log issues if there are any problems or new ideas for the project. The issues might then be opened as a pull request if a change is agreed upon.

4.2 Other possible standard selections

Of the standards discussed in chapter 1, the Center for Internet Security maintained CIS Controls document is very similar to the OWASP ASVS. It is easy to read for a simple developer, and it also features multiple levels for implementations like ASVS. Unlike the ASVS, the CIS Controls are categorised into three groups: basic, foundational and organisational. CIS Control 18 is for ensuring application software security [46]. Most of the controls are for assessing the security of the hardware and software assets as well as security training the personnel. As mentioned in chapter 3, the software supply-chain attacks could also target the personnel or the office hardware and software to access the product being developed there. In this case, CIS Controls would give better protection against possible attacks. Nevertheless, in the case of a web application, like the subject for this thesis, the requirements in CIS Control 18 are not enough to protect against possible cyber attacks.

NIST Cybersecurity Framework serves a similar goal to the CIS Controls document. The framework contains five functions to organise basic cybersecurity activities: identify, protect, detect, respond and recover. The aim of these functions is for the organisation to achieve better risk identification, detection, and management system. As is the case with CIS Controls, the NIST Cybersecurity Framework is excellent for securing against possible phishing attacks or hardware/software intrusions. However, there are few application development specific requirements in the NIST document, which is not enough to assure the security of a web application.

4.3 Security in software lifecycle

As previously discussed in chapter 1, the waterfall model, V-model, and spiral model contain clearly defined steps where analysis and design are completed before any development occurs. These models are very rigid in design and do not allow any deviations from the requirements.

The problem with agile development is that there are no concrete steps for each development stage. Most of the agile methods lack features specifically addressing security risks [47]. Security-related tasks are often ignored as these changes are unnoticeable to the user, and the tasks take much time.

Agile methods instead rely on a trial and error type of iterative approach [48]. The security of an application became added on at the end of the development cycle. Secure Software Development Lifecycle (Secure SDLC) was created to "shift security left". This means that the product's security should be considered as early as possible in the development lifecycle to avoid significant security flaws that are costly to fix.

Secure SDLC addresses security at each stage of development (Figure 15) [49]. The first step of the Secure SDLC is a risk assessment that goes hand-in-hand with requirements analysis. In the architecture and design stage, there should be threat modelling and design review. Threat modelling helps to identify possible threats to the system and how to mitigate them. Design review is there to address common design flaws such as insufficient authorisation or insecure external components. The implementation stage should use static analysis, e.g., a SAST tool, to analyse the source code without executing it to determine if the proper coding convention and standards are followed. The testing stage involves security testing and code review. Security testing tries to find all the loopholes by testing all possible scenarios to find possible vulnerabilities. At the final maintenance stage, there should be a security assessment of possible flaws in the systems that could be improved upon.

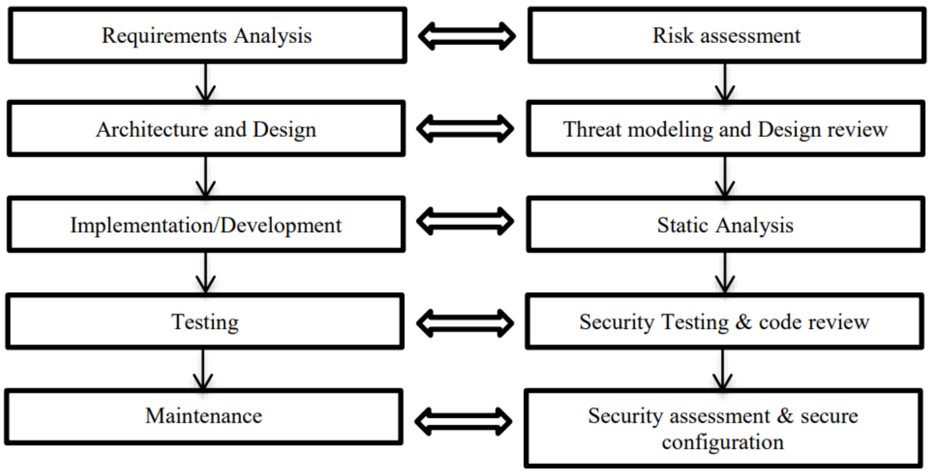

Requirements Analysis
Risk assessment
Architecture and Design
Threat modeling and Design review
Implementation/Development
Static Analysis
Testing
Security Testing & code review
Maintenance
Security assessment & secure configuration

Figure 15 Secure SDLC

For agile software methods, the integrated security methods should be adaptive and straightforward [47]. There should be concrete tools and guidance for all stages of development. The Secure SDLC seems to be a good guide of what steps to take to assure security in all stages of development, which is also the first requirement of the ASVS document.

4.4 Security in software design

The simplest way of assuring application security is to design it with security in mind. In the OWASP Application Security Verification document, the first chapter focuses on the architecture, design and threat modelling requirements [22]. The control objective states that security architecture has become a lost art in many organisations, and the application security field must adopt agile security principles to developers.

To reduce the security vulnerabilities in later stages of development, it would be beneficial to address the risks at the design phase [50]. If the security flaws are removed in the design phase, then it means that the overall software is more secure and efficient.

The first step in design level security is risk assessment. Recognising different security risks at the design stage will help avoid loopholes that might pose a threat in the future. After development, resolving security bugs becomes 100 times more critical than detecting them at a design phase [51].

In 2018, Assal and Chiasson released a report of security practices applied in the software development lifecycle by the industry [52]. They interviewed the developers to explore real-life software security practices used in each software development lifecycle stage. Most of the interviewed indicated that security was not considered as a part of the design stage.

The developers primarily focus on the functional design and dismiss security eighter because they forget to design it or lack the expertise to address the security. A participant event admitted to intentionally introducing complexity to avoid rewriting old code and misusing the frameworks to fit their codebase, dismissing the possibility of introducing vulnerabilities. The result is highly complex code that is hard to understand.

Security is rarely thought of in the design stage of software. When left neglected, it could introduce possible loopholes and security bugs into the code. If the security is considered at the design stage, it would increase integrity and reduce development costs as there would be fewer bugs in the production codebase.

Summary

In the last decade, attacks against software supply-chain have risen in frequency. It is more important than ever to secure the software development lifecycle against possible attacks. This thesis aimed to verify if a security verification standard could protect against software supply-chain attacks.

The thesis was conducted as a case study. The subject of the case study was SimplBooks, an Estonian accounting software. The standard selected for this study was the OWASP Application Security Verification Standard. The standard was selected after conducting a literature review of the existing standards.

For adopting the selected standard, an audit was conducted of the current state of the security of the SimplBooks application and its development process. From the requirements not met, a backlog of tasks was created for future fixes to the application. The existing development process also needed to be complemented by introducing new checklists for the development team.

For verifying the results, the most common software supply-chain attacks in the last decade were identified. The results were that the OWASP ASVS is good enough for securing the application and development process against software supply-chain attacks but lacks the organisational requirements such as employee training about possible cybersecurity attacks.

The main takeaway from this thesis is that ensuring the security of the product should start in the design stage to avoid possible bugs in the future and possible loopholes in the security. When an application is designed with security in mind, it is easier to fix problems and update the security requirements.

References

[1] T. Herr *et al.*, “Broken Trust: Lessons from Sunburst,” 2021. Accessed: Apr. 01, 2021. [Online]. Available: https://www.atlanticcouncil.org/in-depth-research-reports/report/broken-trust-lessons-from-sunburst/.

[2] N. B. Ruparelia, “Software Development Lifecycle Models,” *ACM SIGSOFT Software Engineering Notes*, vol. 35, pp. 8–13, 2010, doi: 10.1145/1764810.1764814.

[3] M. Stoica, M. Mircea, and B. Ghilic-Micu, “Software development: Agile vs. traditional.,” *Informatica Economică*, vol. 17, no. 4, 2013, doi: 10.12948/issn14531305/17.4.2013.06.

[4] W. W. Royce, “Managing the development of large software systems: concepts and techniques,” *Proceedings of the 9th international conference on Software Engineering*, pp. 328–338, 1987, Accessed: Apr. 02, 2021. [Online].

[5] A. Mishra and D. Dubey, “A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios,” *International Journal of Advance Research in Computer Science and Management Studies*, vol. 1, no. 5, 2013, Accessed: Apr. 02, 2021. [Online].

[6] K. Fosberg and H. Mooz, “The Relationship of System Engineering to the Project Cycle,” *INCOSE International Symposium*, vol. 1, no. 1, pp. 57–65, 1991, Accessed: Apr. 02, 2021. [Online].

[7] H. F. Cervone, “Understanding agile project management methods using Scrum,” *OCLC Systems & Services: International digital library perspectives*, pp. 1065–075, 2011, doi: 10.1108/10650751111106528.

[8] “What is Scrum?” https://www.scrum.org/resources/what-is-scrum (accessed Apr. 03, 2021).

[9] J. C. S. Santos, K. Tarrit, and M. Mirakhorli, “A catalog of security architecture weaknesses,” in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Jun. 2017, pp. 220–223, doi: 10.1109/ICSAW.2017.25.

[10] “Microsoft Security Development Lifecycle.” https://www.microsoft.com/en-us/securityengineering/sdl/ (accessed Apr. 03, 2021).

[11] I. Abunadi and M. Alenezi, “An Empirical Investigation of Security Vulnerabilities within Web Applications,” *J. UCS*, vol. 22, no. 4, pp. 537–551, 2016, Accessed: Apr. 03, 2021. [Online].

[12] “OWASP Top Ten 2017.” https://owasp.org/www-project-top-ten/2017/ (accessed Mar. 14, 2021).

[13] P. Sane, “Is the OWASP Top 10 list comprehensive enough for writing secure code?,” in *Proceedings of the 2020 International Conference on Big Data in Management*, Feb. 2020, pp. 58–61, Accessed: Mar. 12, 2021. [Online]. Available: http://arxiv.org/abs/2002.11269.

[14] “CWE - 2020 CWE Top 25 Most Dangerous Software Weaknesses.” https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html (accessed Mar. 12, 2021).

[15] J. Chau, “Application security - it all starts from here,” *Computer Fraud and Security*, vol. 2006, no. 6, pp. 7–9, Jun. 2006, doi: 10.1016/S1361-3723(06)70366-9.

[16] V. v Fomin, H. Vries, and Y. Barlette, “ISO/IEC 27001 information systems security management standard: exploring the reasons for low adoption,” 2008, [Online]. Available: https://www.researchgate.net/publication/228898807.

[17] “ISO - ISO/IEC 27001 — Information security management.” https://www.iso.org/isoiec-27001-information-security.html (accessed Mar. 21, 2021).

[18] J. Task Force and T. Initiative, “Security and privacy controls for federal information systems and organizations,” *NIST Special Publication*, vol. 800, no. 53, pp. 8–13, 2013, doi: 10.6028/NIST.SP.800-53r5.

[19] M. P. Barrett, “Framework for improving critical infrastructure cybersecurity,” *National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep*, Apr. 2018, doi: 10.6028/NIST.CSWP.04162018.

[20] “NIST Cybersecurity Framework vs. NIST Special Publication 800-53 | Praetorian.” https://www.praetorian.com/blog/nist-cybersecurity-framework-vs-nist-special-publication-800-53/ (accessed Apr. 03, 2021).

[21] “The 20 CIS Controls & Resources.” https://www.cisecurity.org/controls/cis-controls-list/ (accessed Mar. 21, 2021).

[22] Open Web Application Security Project, “Application Security Verification Standard 4.0.2,” 2020.

[23] “Annual Report 2019/2020: Shaping the Global Future Together - Atlantic Council.” https://www.atlanticcouncil.org/in-depth-research-reports/report/annual-report-2019-2020-shaping-the-global-future-together/ (accessed Apr. 03, 2021).

[24] “php.internals: Changes to Git commit workflow.” https://news-web.php.net/php.internals/113838 (accessed Apr. 04, 2021).

[25] “PHP: History of PHP - Manual.” https://www.php.net/manual/en/history.php.php (accessed Apr. 04, 2021).

[26] “40 Enterprise Computers Infected with Second-Stage CCleaner Malware - Security Boulevard.” https://securityboulevard.com/2017/09/40-enterprise-computers-infected-second-stage-ccleaner-malware/ (accessed Apr. 04, 2021).

[27] “Inside the CCleaner Backdoor Attack | Threatpost.” https://threatpost.com/inside-the-ccleaner-backdoor-attack/128283/ (accessed Apr. 04, 2021).

[28] "Inside the Unnerving CCleaner Supply Chain Attack | WIRED." https://www.wired.com/story/inside-the-unnerving-supply-chain-attack-that-corrupted-ccleaner/ (accessed Apr. 04, 2021).

[29] "Researchers Link CCleaner Hack to Cyberespionage Group." https://www.vice.com/en/article/7xkxba/researchers-link-ccleaner-hack-to-cyberespionage-group (accessed Apr. 04, 2021).

[30] "Kingslayer - A Supply Chain Attack." https://www.rsa.com/en-us/offers/kingslayer-a-supply-chain-attack (accessed Apr. 04, 2021).

[31] "Flame Exploited Long-Known Flaw in MD5 Certificate Algorithm - Security - News & Reviews - eWeek.com." https://www.eweek.com/security/flame-exploited-long-known-flaw-in-md5-certificate-algorithm/ (accessed Apr. 04, 2021).

[32] "Researchers reveal how Flame fakes Windows Update | Computerworld." https://www.computerworld.com/article/2503916/researchers-reveal-how-flame-fakes-windows-update.html (accessed Apr. 04, 2021).

[33] "Meet 'Flame,' The Massive Spy Malware Infiltrating Iranian Computers | WIRED." https://www.wired.com/2012/05/flame/ (accessed Apr. 04, 2021).

[34] "Operation StealthyTrident: corporate software under attack | WeLiveSecurity." https://www.welivesecurity.com/2020/12/10/luckymouse-ta428-compromise-able-desktop/ (accessed Apr. 04, 2021).

[35] "Lazarus supply-chain attack in South Korea | WeLiveSecurity." https://www.welivesecurity.com/2020/11/16/lazarus-supply-chain-attack-south-korea/ (accessed Apr. 04, 2021).

[36] "Lawmakers press NSA for answers about Juniper hack from 2015 -- FCW." https://fcw.com/articles/2021/01/31/juniper-hack-algo-nsa-letter.aspx (accessed Apr. 04, 2021).

[37] A. Eismont, “A Software Security Assessment using OWASP’s Application Security Verification Standard Results and Experiences from Assessing the DHIS2 Open-Source Platform,” Oslo, 2020.

[38] T. W. Edgar and D. O. Manz, *Research Methods for Cyber Security*. Rockland, MA, UNITED STATES: Syngress, 2017.

[39] “SimplBooks veebipõhine raamatupidamisprogramm on lihtne!” https://www.simplbooks.ee/ (accessed Mar. 14, 2021).

[40] “What is the ICE Scoring Model? | Definition and Overview.” https://www.productplan.com/glossary/ice-scoring-model/ (accessed Apr. 10, 2021).

[41] S. Goswami, N. R. Krishnan, Mukesh, S. Swarnkar, and P. Mahajan, “Reducing attack surface of a web application by open web application security project compliance,” *Defence Science Journal*, vol. 62, no. 5, pp. 324–330, 2012, doi: 10.14429/dsj.62.1291.

[42] F. Ö. Sönmez, “Security qualitative metrics for open web application security project compliance,” *Procedia Computer Science*, vol. 151, pp. 998–1003, 2019, doi: 10.1016/j.procs.2019.04.140.

[43] Open Web Application Security Project, “OWASP Top Ten Proactive Controls Project v 3.0,” 2018. Accessed: Apr. 25, 2021. [Online]. Available: www.owasp.org.

[44] M. Ohm, H. Plate, A. Sykosch, and M. Meier, “Backstabber’s Knife Collection: A Review of Open Source Software Supply Chain Attacks,” doi: 10.1007/978-3-030-52683-2_2.

[45] D. L. Vu, I. Pashchenko, F. Massacci, H. Plate, and A. Sabetta, “Typosquatting and Combosquatting Attacks on the Python Ecosystem,” in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS & PW)*, Sep. 2020, pp. 509–514, doi: 10.1109/EuroSPW51379.2020.00074.

[46] Center for Internet Security, “CIS Controls V7.1,” 2019. [Online]. Available: www.cisecurity.org/controls/.

[47] M. Siponen, R. Baskerville, and T. Kuivalainen, “Integrating security into agile development methods,” in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, pp. 185a–185a, doi: 10.1109/hicss.2005.329.

[48] J. Nguyen and M. Dupuis, “Closing the feedback loop between UX design, software development, security engineering, and operations,” in *Proceedings of the 20th Annual SIG Conference on Information Technology Education*, Sep. 2019, pp. 93–98, doi: 10.1145/3349266.3351420.

[49] R. K. Gundla, “Secure Software Development Lifecycle,” in *International Journal of Scientific Development and Research*, 2018, vol. 3, Accessed: May 09, 2021. [Online]. Available: www.ijsdr.org.

[50] J. Kaur and R. Ahmad Khan, “Major Software Security Risks at Design Phase,” *ICIC Express Letters*, vol. 12, no. 11, pp. 1155–1162, 2018, doi: 10.24507/icicel.12.11.1155.

[51] B. Boehm and V. R. Basili, “Software defect reduction top 10 list,” *Foundations of empirical software engineering: the legacy of Victor R. Basili*, vol. 426, no. 37, pp. 426–431, 2005.

[52] H. Assal and S. Chiasson, “Security in the software development lifecycle,” *Fourteenth Symposium on Usable Privacy and Security ({SOUPS} 2018)*, pp. 281–296, 2018, Accessed: May 10, 2021. [Online]. Available: https://www.usenix.org/conference/soups2018/presentation/assal.