Chapter 1

Introduction

1.1 Background

Throughout the last ten years, mass cyberattacks against major organizations have amplified. Security breaches are the most prominent cause for attacks being allowed to happen. Although different types of organizations become victim of cyberattacks, an analysis of data breaches experienced in multiple organizations established that medical organizations and BSOs are the least prepared against attacks [1]. The latest reports confirm that vulnerabilities continue to rise as a result of the pandemic, and almost half of all businesses have been confronted with cyberattacks since companies are settling into the new normal [2]. Cyberattacks have grown in frequency and severity since the pandemic. Lallie, Harjinder Singh et al observed that there appears to be a loose correlation between the announcement and a corresponding cyber-attack campaign that utilizes the event as a hook to increase the likelihood of success [3]. Though solutions to counter cyberattacks exist, guidance on implementing software security is needed.

Security breaches are caused by security vulnerabilities in source code introduced by software developers when creating software, and therefore developers are often blamed for vulnerabilities [4]. However, application security is primarily performed by security experts causing a separation between security and devel-

opment. As a result, the probability of insecure software is increased [5].

Writing secure code therefore is critical with the prevalence of security vulnerabilities. To achieve this, developers need to be aware of the potential vulnerabilities they might introduce when developing software features and understand how to mitigate them. Still, the knowledge and skills to produce secure software are lacking and often are not taught in computing curricula despite the existence of secure coding guidelines [6] [7] [8] [9].

Such secure code guidelines are provided by the Open Web application Security Project (OWASP). Security standards such as the OWASP Application Security Verification Standard (ASVS) and the Mobile Application Security Verification Standard (MASVS) exist to guide organizations and developers to produce secure software through categorized security controls that can be implemented during the development lifecycle. Furthermore, OWASP has cheat sheets and a series of security testing guides that provides test cases that verifies the security controls put in place. Organizations that are serious about security, should apply security standards and security testing guides during the Software Development Life Cycle (SDLC). The SDLC is a process used to plan, define requirements, design, implement, test and deploy software. It ensures that software development is done reliable, cost-effective in a given time window. Implementing security in the SDLC is the first step in assuring secure software is produced. However, developers might not find it easy to work with such guidelines [10].

The OWASP Security Knowledge Framework (SKF) is security expert system that uses secure code guidelines such as the OWASP ASVS and OWASP MASVS to assist developers during the SDLC. To address security during the SDLC, this study will implement a secure SDLC (SSDLC) with SKF for the development of CheFeed, a full stack mobile application developed for this thesis as group. CheFeed is a social recipe sharing application that also applies sentiment analysis on recipe reviews. Though CheFeed is minimal in its functionality, features such as user authentication and user session management needs to ensure it follows

security standards. Thus, CheFeed is an appropriate candidate for implementing a SSDLC.

1.2 Aims and Benefits

1.2.1 Aim

Security vulnerabilities found in applications introduced by software developers are the cause of many security breaches and data theft. To reduce the introduction of security vulnerabilities security must be addressed during software development. The MASVS has categorized security controls that can be used as guidance to develop secure software. In addition, it maps security controls to test cases that can be found in the OWASP Mobile Security Testing Guide (MSTG). However, implementing security standards such as the MASVS might be a difficult task. Without a proper background in security, a developer might not know where to start and understand what security controls are relevant to the project. Therefore, this study aims to implement SKF in an agile SSDLC where security activies are performed based on MASVS security controls defined for feature sprints.

1.2.2 Benefits

This study will provide insights into the process of developing software in an agile SDLC where security is addressed from the beginning. It provides a clear presentation of the benefits of using SKF to configure feature sprints where security should be in place. Students and software developers may benefit from understanding how secure code guidelines can be used. Furthermore, the implementation of SKF for a secure SDLC could serve as a tool for future studies to build upon the process presented in this study.

1.3 Scope

The MASVS and other security standards understands that not all applications are equal in terms of its potential for security vulnerabilities. Thus, several security verification levels exist. The scope of this study is limited to MASVS *Level 1* (MASVS-L1). MASVS-L1 ensures that mobile applications cohere to the best security practices concerning code quality, handling of sensitive data, and interaction with the mobile environment. Security activities are performed during the *requirements, design, implementation*, and *testing* phases of the SDLC.

CheFeed is a full stack mobile application that has been developed as a final group project for our undergraduate thesis. The overall scope for the development of the project involves the development of the REST API, the development of a sentiment analysis model that will be served on the REST API processing recipe reviews from users, and the development of the client side. The focus of each contributor outside this study is delegated as follows:

- The development of the API and database design by Stephanus Jovan Novarian in his thesis "CheFeed: Development of CRUD backend services on recipes".
- The development of a sentiment analysis model which uses recipe reviews as its input by Ikshan Maulana in his thesis "CheFeed - The Implementation of different RNN Architectures".

Chapter 2

Theoretical Foundation

The objective of this chapter are to introduce fundamentals of information security and software engineering to provide a framework for understanding the rest of this study. First a definition of information security is given. Then concepts such as the *confidentiality*, *integrity*, and *availability* (CIA) triad as well as the *Gold Standard* are discussed. Lastly, software engineering activities together with software security are presented.

2.1 Information Security

This section defines information security. Moreover, the section will discuss how the evolution of computers and the World Wide Web has introduced challenges to information security. Lastly this section defines what secure by design means.

2.1.1 Fundamentals of Information Security

It is important to understand what precisely is implied when discussing security. Therefore, we have to properly define it. For instance, the terms cybersecurity and information security are commonly used indistinguishably. The Cambridge Dictionary defines information security as "methods used to prevent electronic information from being illegally obtained or used", and defines cybersecurity as "things that are done to protect a person, organization, or country and their

computer information against crime or attacks carried out using the internet". The definition for cybersecurity describes a much larger scope for security.

Solms et al [11] supports this argument and reasons that information security is solely about securing the information, generally referred to as the asset, from potential threats posed by inherent vulnerabilities. Furthermore, they outlined that cybersecurity goes beyond protecting assets. Cybersecurity includes the insurance of those that operate in cyberspace in addition to any of their assets that can be achieved through cyberspace. Although the definition of information security and cybersecurity overlap each other, the latter is much more extensive in its definition. Overall, security is about securing assets against the most probable types of attacks, to the best ability [12].

Information security can be defined as the protection of information from potential abuse after various threats and vulnerabilities [11]. In a general sense, security means protecting our data and systems assets from whoever intends to misuse them. It includes several aspects of business, involving financial controls, human resources, and protection of the physical environment, as well as health and safety measures [13]. Security strives to secure ourselves against the most likely forms of attack, to the best ability.

Security is built on top of well-established principles like the Confidentiality, Integrity and, Availability (CIA) triad and the gold standard.

The Confidentiality, Integrity, and Availability Triad

Security can be complicated as discussed in Section **??**. Nonetheless, as described in Table 2.1, the CIA triad, provides a model to think about and discuss security concepts. It is commonly discussed in the information security literature [12] [14] [15]. The CIA triad is commonly referred to as tenets of information security. Information assets that are tied to an application can be associated with a specified CIA requirement represented as a number or values suchlike, high, medium, or low, which can be determined through risk analysis [14].

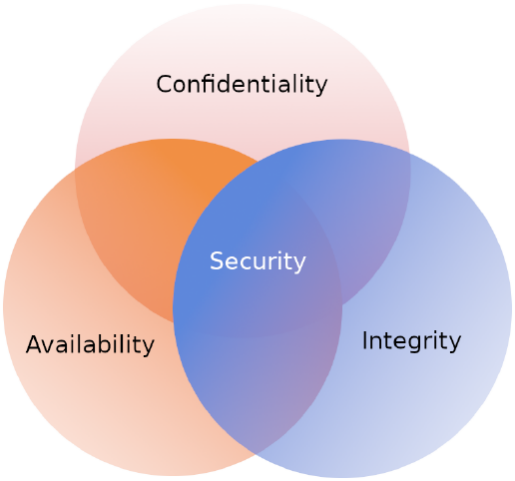


Figure 2.1: The CIA Triad

Confidentiality Confidentiality describes the capability to secure assets from parties that do not have the authorization to view them.

Integrity The second concept describes the ability to prevent data from being changed in an unauthorized or undesirable manner. Integrity preserves the consistency of information both internally and externally.

Availability The final concept describes the ability to access data when required.

The Gold Standard

Authentication Identification is the claim of identity by a person, process, or other entity without implying the authenticity of the claim or privileges that could be affiliated with the identity. Many methods exist to claim our identity such as name abbreviations, fingerprints, portraits, and many more.

Authentication is the procedure used to validate whether the claim of identity is correct. A real-world example of authentication would be the usage of a username and password combination inside an application. Depending on the security level required of an asset, more factors can be used for the authentication mechanism, also known as multifactor authentication.

Authorization Besides claiming an identity and confirming the validity of that claim, we need to decide what the party is allowed to do and if access to specific resources is allowed or denied.

Principle of least privilege An important authorization concept is the principle of least privilege. It mandates that only the bare minimum of access to a party should be allowed to function. As an example, a user account is only granted the access needed to perform their routine work. It is a very simple security measure that requires minimal effort, and it is highly effective.

Access control At a high level, access control is about restricting access to a resource. Access control can be divided into two groups to either improve the design of physical security or cybersecurity. Generally, four basic actions can be performed:

- allowing
- access
- denying access
- limiting access
- revoking access

Most access control issues or situations can be described through these. Moreover, users that are not authorized should not be granted access. Therefore, it is best practice to disallow access by default.

Two main methods can be considered to implement access controls: access control lists (ACLs) and capabilities. ACLs often referred to as "ackles", are a very common choice of access control implementation. Typically, ACLs are implemented in the file systems on which our operating systems run and to control the flow of traffic in the networks to which our systems are connected. A capability-based approach to security uses tokens that manage our access. A good analogy

would be the usage of a personal badge that grants access to certain doors inside a building. Notably, the right to access a resource is based completely on possession of the token, not who possesses it.

Auditing After going through the process of identification, authentication, and authorization, it is important to keep track of the activities that have occurred. Despite access being granted to the party, it is important that the party behaves according to the rules as it concerns security, ethics, business conduct, and so on. With an abundance of digital assets it has become a vital task to ensure that rules set forth are abided by.

Cryptography

Cryptography is the science of ensuring that assets are kept secure. The foremost security measure allowing cryptography is encryption, and often the terms are used interchangeably. Although in reality, encryption is a subset of cryptography. Encryption is the transformation of plaintext into ciphertext.

Cryptology is not a recent invention. At the very least cryptology can be traced back as far as 2500 years and was considered an obscure science. It was well established with both ancient Greeks and Romans who practiced different forms of cryptography. A classic example of ancient cryptography is the Ceasar cipher as seen in Figure 2.2. After the fall of the Roman Empire, cryptology was flourishing in the Arabic world [16].

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Figure 2.2: Ceasar Cipher

Without cryptography, much of the internet-based activities we benefit from today would be at great risk. Cryptography is essential in computing, networking, and the great set of transactions that take place over such devices in everyday life. Cryptography has permitted us to become a very network-centric society.

Data can be protected at rest, in motion, and to a certain extent, in use, because of cryptography. Thus allowing us to securely communicate and perform transactions when sensitive data is involved.

The process of encrypting plaintext and decrypting ciphertext is described as a cryptographic algorithm. To either encrypt or decrypt a message, cryptographic algorithms commonly use a key, or multiple keys, with a range of possible values for the key referred to as the keyspace. The harder the keyspace, the harder it is to decrypt the message. We will take a brief look at some popular cryptographic algorithms.

2.1.2 Symmetric cryptography

Symmetric cryptography, also referred to as private key cryptography, utilizes a single key for both encryption of the plaintext and the decryption of the ciphertext as can be seen in Figure **??**. A symmetric cipher only works if both the sender and the receiver own same key to unlock the cipher. Therefore, everyone who uses a symmetric cipher must have the same set of keys and must use them in the correct order [16].

2.1.3 Asymmetric cryptography

When a different key is used for encryption and decryption, we have an asymmetric system in place. Asymmetric cryptography can also be referred to as public key cryptography. Asymmetric cryptography relies on a public key to encrypt data from the sender, and a private key to decrypt data that arrives at the receiving end as seen in Figure **??**. Due to the mathematical complexity of the operations to create the private and public keys, no method exists at present to reverse the private key from the public key.

2.1.4 Hash functions

Unlike both symmetric and asymmetric cryptography, which relies on keys for encryption and decryption, some algorithms do not require keys, known as hash functions. Hash functions generate a generally unique and fixed-length hash value, referred to as a hash, based on the original message as seen in Figure **??**. Any form of change to the message will change the hash as well. Furthermore, hash functions do not allow for the contents of the message to be read, though they can be utilized to determine the confidentiality of the message. Some hash algorithms include Message-Digest 5 (MD5), MD2, MD4, SHA-2, and RACE.

Digital signatures A good example of where hash functions are utilized is digital signatures. To detect any changes to the content of the message, digital signatures make it possible to sign a message to ensure the authenticity from the sending party. This is accomplished by generating a hash of the message, and then using the senders private key to encrypt the hash, thereby creating a digital signature. The receiving party can use the senders public key to decrypt the digital signature, thereby restoring the original hash of the message.

Digital signatures are now recognized as legally binding in many countries, allowing them to be used for certifying contracts or notarizing documents, for authentication of individuals or corporations, as well as components of more complex protocols. Broadly speaking, a digital signature is analogous to a handwritten signature, which provides much stronger security guarantees [17].

Certificates Another form of cryptography for message signing is the usage of digital certificates, commonly known as certificates. Certificates link together a public key and an individual, typically by taking the public key and something to identify the individual, suchlike a name and address, and having them signed by a certificate authority (CA). A CA is a trusted entity that is responsible for digital certificates. The advantage of using a certificate is that it verifies that a public key is associated with a particular individual.

2.2 Software Engineering

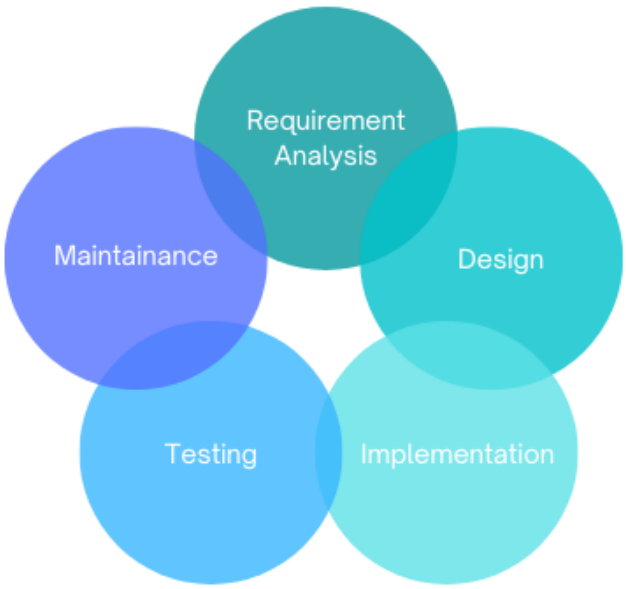
Software is embedded into today's society, and it ranges from simple embedded systems to complex, worldwide information systems. There is no natural limit to the potential of software. However, because of this freedom, software can easily become highly complex, difficult to comprehend, and costly to change. Individual approaches to software development did not scale up to large and complex software systems. Thus, the notion of software engineering was proposed in 1968. Software engineering aims to assist professional software development rather than individual programming. It is an engineering principle concerned with all aspects for the creation of software [18].

The SLDC is a process with related activities that leads to the production of a software system. In this section, the SDLC is presented and what common security vulnerabilities are found in deployed software. Lastly, the role of security in the SDLC is discussed.

2.2.1 The Software Development Life Cycle

The purpose of the SDLC has adjusted over time since the first concept was presented by Herbert Benington in 1956. Initially, the concept of the SDLC was about understanding what needed to be done. Later, *sequential* SDLC models were introduced and had strongly controlled development activities. The growing focus on processes, strong control, tool support and so forth led to a counter-movement advocating for self-organization, iterative life cycles and suchlike. As a result, *agile* methodologies and the parallel plan-driven and the agile cultures grew in importance. Today, it is understood that both the sequential and agile approach to software development have their pros and cons [19]. Though a variety of SDLC models exist, an overview of the SDLC phases is shown in figure 2.3.

Figure 2.3: The SDLC phases



The Waterfall Model

The Agile Model

The Spiral Model

2.2.2 Security in the Software Development Life Cycle

Common Vulnerabilities

The OWASP Top Ten

Broken Access Control

Security Engineering

Bibliography

[1] Hicham Hammouchi et al. "Digging Deeper into Data Breaches: An Exploratory Data Analysis of Hacking Breaches Over Time". In: *Procedia Computer Science* 151 (2019), pp. 1004–1009. DOI: 10.1016/j.procs.2019.04.141. URL: https://doi.org/10.1016%2Fj.procs.2019.04.141.

[2] Charles Weir et al. "Infiltrating security into development: exploring the world's largest software security study". In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, Aug. 2021. DOI: 10.1145/3468264.3473926. URL: https://doi.org/10.1145%2F3468264.3473926.

[3] Harjinder Singh Lallie et al. "Cyber security in the age of COVID-19: A timeline and analysis of cyber-crime and cyber-attacks during the pandemic". In: *Computers amp Security* 105 (June 2021), p. 102248. DOI: 10.1016/j.cose.2021.102248. URL: https://doi.org/10.1016%2Fj.cose.2021.102248.

[4] Hala Assal and Sonia Chiasson. "*less*iThink secure from the beginning*less*/i". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, May 2019. DOI: 10.1145/3290605.3300519. URL: https://doi.org/10.1145%2F3290605.3300519.

[5] Tyler W. Thomas et al. "Security During Application Development". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing*

Systems. ACM, Apr. 2018. DOI: 10.1145/3173574.3173836. URL: https://doi.org/10.1145%2F3173574.3173836.

[6] Madiha Tabassum et al. "Evaluating Two Methods for Integrating Secure Programming Education". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, Feb. 2018. DOI: 10.1145/3159450.3159511. URL: https://doi.org/10.1145%2F3159450.3159511.

[7] Huiming Yu et al. "Teaching secure software engineering: Writing secure code". In: *2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR)*. IEEE. 2011, pp. 1–5.

[8] Tiago Espinha Gasiba et al. "Is Secure Coding Education in the Industry Needed? An Investigation Through a Large Scale Survey". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, May 2021. DOI: 10.1109/icse-seet52601.2021.00034. URL: https://doi.org/10.1109%2Ficse-seet52601.2021.00034.

[9] Daniela Seabra Oliveira et al. "API Blindspots: Why Experienced Developers Write Vulnerable Code". In: *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 315–328. ISBN: 978-1-939133-10-6. URL: https://www.usenix.org/conference/soups2018/presentation/oliveira.

[10] Yasemin Acar et al. "Developers Need Support, Too: A Survey of Security Advice for Software Developers". In: *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, Sept. 2017. DOI: 10.1109/secdev.2017.17. URL: https://doi.org/10.1109%2Fsecdev.2017.17.

[11] Rossouw von Solms and Johan van Niekerk. "From information security to cyber security". In: *Computers Security* 38 (Oct. 2013), pp. 97–102. DOI: 10.1016/j.cose.2013.04.004. URL: https://doi.org/10.1016%2Fj.cose.2013.04.004.

[12] Jason Andress. *The basics of information security : understanding the fundamentals of InfoSec in theory and practice*. Waltham, MA: Syngress, 2014. ISBN: 0128007443.

[13] Leron Zinatullin. *The psychology of information security : resolving conflicts between security compliance and human behaviour*. Ely, Cambridgeshire: IT Governance Publishing, 2016. ISBN: 1849287899.

[14] M. L. Srinivasan. *CISSP in 21 days : boost your confidence and get the competitive edge you need to crack the exam in just 21 days*. Birmingham, UK: Packt Publishing, 2016. ISBN: 9781785884498.

[15] Darren Death. *Information security handbook : develop a threat model and incident response strategy to build a strong information security framework*. Birmingham, UK: Packt Publishing, 2017. ISBN: 9781788478830.

[16] John Dooley. *History of cryptography and cryptanalysis : codes, ciphers, and their algorithms*. Cham, Switzerland: Springer, 2018. ISBN: 9783319904436.

[17] Jonathan Katz. *Digital signatures*. New York: Springer, 2010. ISBN: 0387277110.

[18] Ian Sommerville. *Software Engineering*. 10th ed. Pearson, 2021. ISBN: 9780133943030,1292
URL: http://gen.lib.rus.ec/book/index.php?md5=EB8FF24BBA604674DD15D3E029CA

[19] Ralf Kneuper. "Sixty Years of Software Development Life Cycle Models". In: *IEEE Annals of the History of Computing* 39.3 (2017), pp. 41–54. DOI: 10.1109/mahc.2017.3481346. URL: https://doi.org/10.1109%2Fmahc.2017.3481346.