CHEFEED

DESIGNIN SECURE MOBILE APPLICATIONS WITH OWASP SECURITY KNOWLEDGE FRAMEWORK

THESIS

Prepared by

Imanuel Febie 2201835800

Approved by:

Supervisor

Raymond Bahana

BINUS UNIVERSITY

Jakarta

2022

Abstract

Application security grows more and more important as both web and native applications are becoming more prevalent. It is not uncommon for sensitive data being exposed due to insecure code. The latest Open Web Application Security Project (OWASP) Top Ten release revealed that injection vulnerabilities still ranks as number three, implying that software developers do not properly know how to mitigate these vulnerabilities or are simply not aware of these. Broken access control and cryptographic failures takes the first and second place respectively on the list. However, this problem can be solved by taking application security seriously and adding it into the software development life cycle. This thesis proposes adding the OWASP Security Knowledge Framework to the software development life cycle through a case study.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Are software developers responsible for the cyberattacks that has increased over the last decade? An analysis of over 9000 data breaches since 2005 have revealed that data breaches have contributed to the loss of 11,5 billion individual records with major financial and technical impact [1]. Countless attacks are allowed to happen as a consequence of software vulnerabilities that leave web applications, web servers, or websites exposed. Software developers increase the risk of a security vulnerability each time a new feature is added to an application, albeit many can be mitigated with a secure-by-design approach. However, software developers need to have adequate knowledge on how to mitigate common security vulnerabilities. Ergo the knowledge and skill to develop secure software needs to be taught to software developers, yet is commonly overlooked [2].

For this thesis a hybrid mobile application will be developed using FastAPI, a python framework to develop the Application Programming Interface (API), and React Native, a JavaScript library to develop User Interfaces. The mobile application will be a social recipe sharing called CheFeed.

A well known source for best practices in software security is the Open Web Application Security Project (OWASP). OWASP is a nonprofit foundation that concerns itself on improving the security of software, and has over 250 local chapters around the globe. It includes community-led open source software projects [3]. This thesis proposes to implement the Security Knowledge Framework (SKF) into an agile Software Development Life Cycle (SDLC). SKFs objective is to train developers and teams in writing secure code using the Application Security Verification Standard (ASVS) and Mobile Application Security Verification Standard (MASVS).

1.2 Scope

This study covers the implementation of SKF for developing secure mobile applications. SKF provides three security maturity levels from ASVS and MASVS. This study limits itself to achieve level 1 for security for the features users and authentication. In practice, a security risk assessment or threat modeling should be added to a secure software development cycle. Thefore, the thesis will limit itself to:

- gathering security requirements with SKF for chefeed user authentication and session managements
- development of the authentication API
- development of the client side
- security verification

CHEFEED is a group effort and the scope and responsibilities outside this thesis are described in Table 1.1.

Table 1.1: Other member scope overview

Member	**Scope**	**Thesis**
Ikhsan Maulana	Sentiment Analysis, back-end development	Sentiment Analysis on Food Reviews
Stephanus Jovan Novarian	SDLC, back-end development	Implement Agile Software Development Life Cycle on CHEFEED

1.3 Aims and Benefits

1.3.1 Aims

Software code is the essence of each application. Applications that we use daily are processing important and sensitive assets. Therefore, the confidentiality, integrity, and availability of those assets are at risk of being compromised. However, adding security might be a challenging task. The aim of this thesis is to demonstrate how security can be added to the SDLC by constructing sensible security requirements for CHEFEED. The security requirements will be based on the OWASP ASVS and MASVS by utilizing OWASP SKF.

1.3.2 Benefits

Secure programming forces developers to improve coding standards as well as the overall coding architecture. Adding security to the SDL will save development time, since there is no need to add new code after security audits are performed.

1.4 Structure

This section summarizes the general description of what is discussed in each chapter of the thesis.

Chapter 1 - Background Chapter 1 introduces the main topics of the thesis through the background section, and it defines the scope. Lastly, it describes the aims and benefits.

Chapter 2 - Theoretical Foundation Chapter 2 starts with defining security and introduces important information security concepts, software development and security testing.

Chapter 3 - Analysis on Secure Software Development Chapter 3 analyzes several solutions to develop secure software with well established standards, processes and guides. Lastly it will introduce the proposed solution and presents the security requirements for developing CheFeed.

Chapter 4 - Security Solution Design Chapter 4 discusses the solution designed based on the requirements discusses in Chapter 3.

Chapter 5 - Testing Chapter 5 discusses the security tests performed for CheFeed.

Chapter 6 - Discussion Chapter 6 outlines the authors experience and thoughts on the proposed solution.

Chapter 7 - Conclusion and Recommendation The final chapter concludes the thesis and provides further recommendations.

Chapter 2

Foundation

Chapter 2 discusses the theoretical foundations and frameworks related to security, software development and security testing. First a definition of information security is described, followed by the challenges that are faced in security and what "secure by design" means. Furthermore, this chapter dives into the fundamentals of security such as the confidentiality, integrity, and availability triad, the gold standard, and cryptography. Besides security, it discusses software development as well the SDLC and security from the software engineers perspective. Lastly, this chapter discusses security testing.

2.1 Information Security

This section defines information security, specifically on how it is different from cybersecurity. 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 Defining Information Security

It is critical 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 [4] 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 [5].

Information security can be defined as the protection of information from potential abuse subsequent from various threats and vulnerabilities [4]. In a general sense, security means protecting our data and systems assets from whoever who intends to misuse it. It includes several aspects of business, involving financial controls, human resources and protection of the physical environment, as well as health and safety measures [6]. Security strives to secure ourselves against the most likely forms of attack, to the best ability.

2.2 Fundamentals of Information Security

Security is built on top of well established principles like the Confidentiality, Integrity and Availability (CIA) triad and the gold standard. This section discusses those principles as well as the role of cryptography. In addition, the software development life cycle is discussed and the technologies and languages.

2.2.1 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 [5] [7] [8]. The CIA triad is commonly referred to as tenets of information security. Information assets that are tied to an application can be associated to a specified CIA requirement represented as a number or values suchlike, high, medium, or low, which can be determined through risk analysis [7].

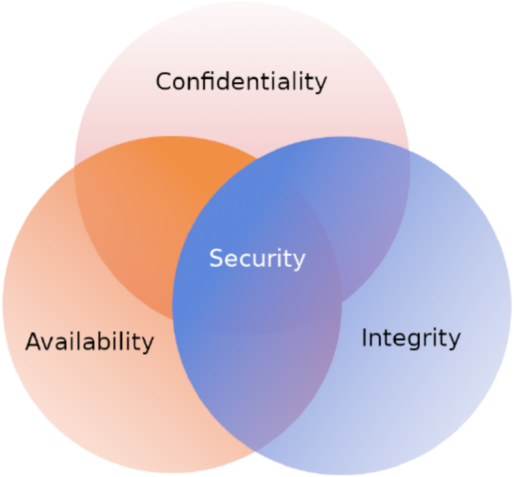


Figure 2.1: The CIA Triad

Confidentiality

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

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.

2.2.2 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 exists 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 are 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.

There are two main methods that 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 manages 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.

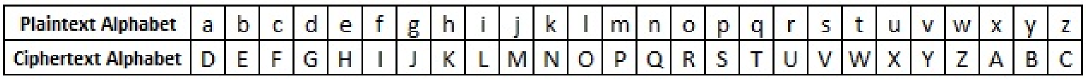
Auditing

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 to 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.

2.2.3 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 [9].



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. In fact, 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. In order 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.

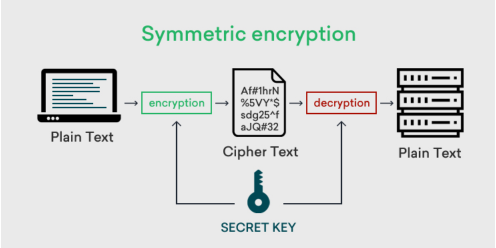


Figure 2.3: Symmetric encryption

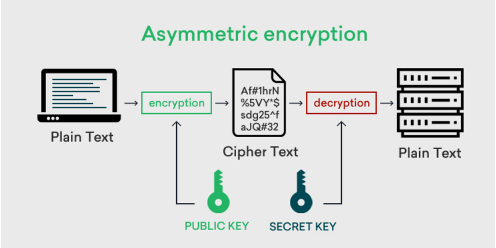


Figure 2.4: Asymmetric encryption

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 2.3. A symmetric cipher only works if both the sender and the receiver are in possession of the 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 [9].

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 2.4. Due to the mathematical complexity of the operations to create the private and public keys, no method exist at present to reverse the private key from the public key.

Hash functions

Unlike both symmetric and asymmetric cryptography, which relies on keys for encryption and decryption, there are algorithms that 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 2.5. Any form of change to the message will change the hash as well. Furthermore, hash functions do not allow for contents of the message to be read, though it can be utilized to determine the confidentiality of the message. Some hash algorithms include: Message-Digest 5 (MD5), MD2, MD4, SHA-2, and RACE.

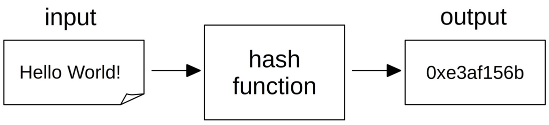


Figure 2.5: Hash function

Digital signatures A good example of where hash functions are utilized are 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 use the senders private key to encrypt the hash, thereby creating a digital signature. The receiving party can use the sender's 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, that provides much stronger security guarantees [10].

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 provides verification that a public key actually is associated with a particular individual.

2.3 Software Development

Software development has become more complex over the years. Today software is dependent on multiple components. More components also introduce a greater risk to software vulnerabilities. This section dives into modern software application architecture, technology stacks relevant to this thesis and attack vectors software engineers should be aware about.

2.3.1 Software Architecture

Every application has design decision in place, whether implicit or explicit. Design decisions include all the aspects of the system under development, including structure, functional behavior, nonfunctional properties, user interaction, and decisions associated to the software's implementation and deployment [11]. Software architectures often are described with diagrams using boxes, circles, and lines accompanied with writing.

The Three-tier Architecture

2.3.2 Technology Stack

Containerization

Docker describes a container as "*a standard unit of software that packages up code and all its dependencies so the application runs quicly and reliably from one computing environment to another*" [12]. For many years, Linux distributions have included containers. However, due to its complexity they have rarely been used. The introduction of Docker unlocked the value of Linux containers by combining an easy to use standardized packaging format. Processes that once were abstract have become comprehensible for developers and operation teams. With Docker the process of creating a distributable product for any application, deploying it at scale into any environment is no longer complex. Docker has greatly simplified the workflow and responsiveness of agile software organizations.

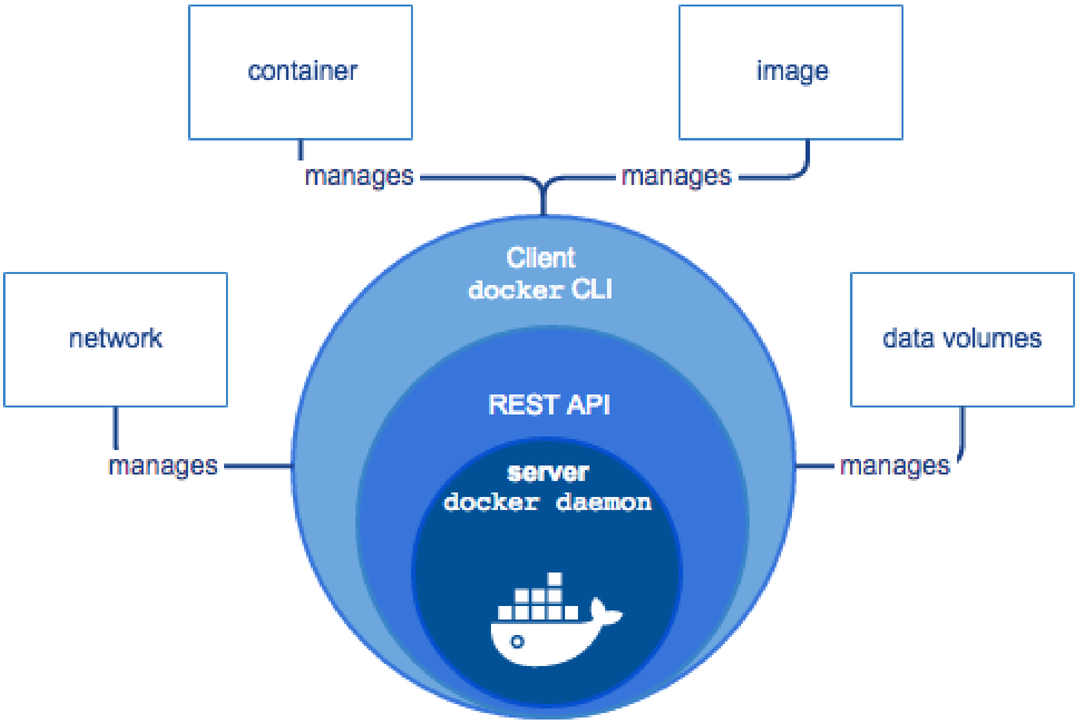


Figure 2.6: Docker Overview

When implemented correctly, any organization, team, developer or operation team will benefit from the use Docker [13].

NoSQL Databases

Application Programming Interface

The Client

2.3.3 Software Attack Vectors

2.3.4 The Software Development Life Cycle

The SDLC plays an essential role for software engineers since it assists in defining the foundation to develop software. Even as early as the 1950s there was the need to develop methodologies for software development in order to accelerate software development. Although the complexity of software has increased, the procedure of the development life cycle should in essence stay the same whilst the implementation to each project should be tailored respectively. In practice, two important questions arise in software development [14]:

1. Is the problem identified correctly?
2. What is the proper software solution?

The SDLC ensures that complex reliable software can be designed and developed cost-effective within the given time. Often this process is described as the SDLC model [15].

Software Development Life Cycle Models

In 1956, the first definite representation of an SDLC model was introduced by Herbert Benington. He introduced the waterfall model that describes software development as a cascading waterfall, that flows from top to bottom as seen in Figure 2.7. This framework for software development introduced several advantages suchlike its ease to understand. However, the waterfall model also has disadvantages. For instance, working software is not available until late during the life cycle, and it is also an inadequate model for large projects [15]. The models begins with an incomplete implementation of a total system followed by increased functionality and performance over time assuming that all requirements are understood properly [16]. The central question SDLC models tried to answer in the early days was how to develop software at all, and what steps would be required. Consequently, the first SDLC models were sequential models such as the waterfall model [17].

Today many SDLC models exist such as the Iterative Model, Spiral Model, V-Model, Big Bang Model, Agile Model, Software Prototype and Rapid Application Development Model. Unnati S. Shah et al observed that though many SDLC models exist, they are complementary, not competitive. Furthermore, they grouped all models into three broad categories; Traditional models, Agile models and Hybrid models [16].

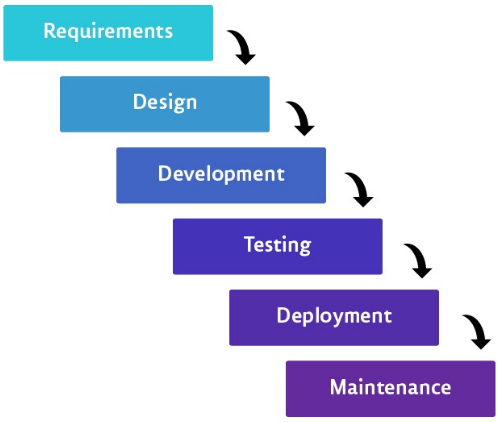


Figure 2.7: Software Development Life Cycle - Waterfall Model

The objective of an SDLC model is to provide a structure for the different software development activities. Albeit many models exist for the SDLC, it is generally divided in several phases including planning, defining requirements, design of software the architecture, development, and testing. A brief description of each phase is described below.

Planning Planning provides the foundation of every SDLC. The objective of the planning phase is to acquire the quality assurance, risk identification and feasibility report.

Defining Requirements The next phase is centered on defining the details of the requirements gathered during the planning phase, document those and receive verification from the customer. The objective is the software development specification (SRS) document which comprises all the requirements of the product to be designed and developed.

Designing the Software Architecture Before actual development is started, the SRS is used as input to design several architectures of the software product. After reviewing the architectures the best design is selected.

Development of the Product Based on the design of the previous phase, the actual development of the software product can begin.

Testing The final phase tests the developed software to assess whether it meets the user requirements defined in the SRS. If the tests are successful, the product can be deployed.

Agile Software Development

The "Agile Manifesto" introduced agile software development (ASD) in 2001 to the software engineering community. Described in the manifest was its purpose and principles [18]. The four core values described in the manifesto included:

- Individual and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

ASD does not assume that all requirements are clear before implementation unlike the traditional waterfall approach. As stated in the Agile Manifesto, "We've examined plenty of successful projects and few, if any, delivered what was planned in the beginning, yet they succeeded because the development team was agile enough to respond again and again to external changes" [18]. Instead of trying to reject the inevitable changing nature of requirements it focuses on managing the changing nature of requirements through the SDLC as illustrated in 2.8. Therefore, agile requires a close relationship with business expertise [16].

2.3.5 Software Security Engineering in the Software Development Life Cycle

Secure can be defined as "freedom from risk and the threat of change for the worse". From the software engineers perspective security is about engineering software such that assets are free from risk and the threat of change for the worse, or at least to the best ability. Secure programming is about designing

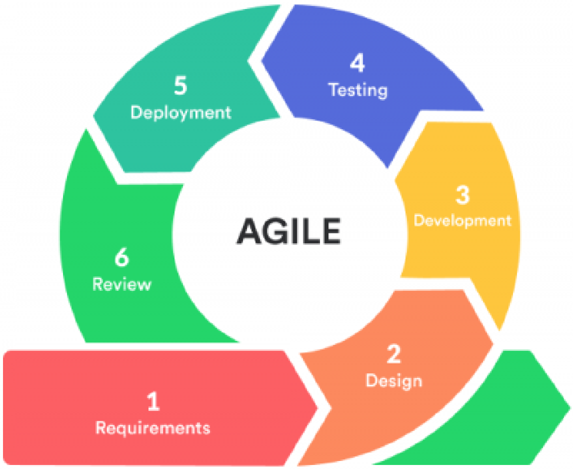


Figure 2.8: Agile Development Flow

and implementing software with the minimal amount of vulnerabilities that an attacker can exploit [19]. However, modern software is complex and fragile. Despite professional engineers being capable of testing and debugging code, security is a different issue, because insecure code generally works without issues, given no attacker is exploiting the code. Software Security Engineering (SSE) introduces processes to ensure software security is ensured throughout the SDLC.

Risk Assessment

Threat Modeling

2.4 Security Testing

Security testing is about ensuring that the software being developed behaves properly when the software is intentionally being abused, and thus the security controls operate as expected. Therefore, potential risks must be identified and their impact. In a 2004 paper by Potter B, and McGraw G, the importance of security testing was already emphasized [20]. In summary security testing involves two diverse approaches:

- security mechanisms such as cryptography, strong authentication, and access control should be tested to ensure their functionality is correctly implemented, and

- risk-based security testing encouraged by understanding and simulating the attackers approach should be performed.

Indeed, performing security test is important. However, the question then is; when should security tests be performed? Too often security and security tests are still an afterthought. In reality, this is ineffective and expensive. To prevent security bugs from even appearing in production is to enhance the SDLC and integrate security into each phase [21].

Chapter 3

Analysis on Secure Software Development

3.1 Challenges in Software Security

3.2 Existing Solutions

3.2.1 The OWASP Top Ten

3.2.2 The OWASP Application Security Verification Standard

3.2.3 The OWASP Mobile Application Security Verification Standard and the Mobile Security Testing Guide

The MASVS similar to the ASVS provides a security standard to design, develop and test secure mobile apps. Although smartphones are small computers running on an operating system with mobile apps, the security requirements are not necessary similar. Data such as our personal information, pictures, recordings, notes, account data, business information, location and much more are stored on mobile apps. Therefore, mobile security is centered around data protection [22].

MASVS Verification Levels

Unlike the ASVS The AppSec model for mobile applications defines two security verification levels. However, in addition to the two security verification levels the MASVS adds a set of reverse engineering resiliency requirements.

- **MASVS-L1**: Standard Security
- **MASVS-L2**: Defense-in-Depth
- **MASVS-R**: Resiliency Against Reverse Engineering and Tampering

The MASVS Level 1 (L1) achieves basic requirements and complies to mobile application security best practices. Testing is required to validate the security controls. All mobile applications are recommended to implement L1.

3.2.4 The OWASP Security Knowledge Framework

OWASP SKF aims to train and guide software developers in building secure applications, by design. It is an open source web application that can be accessed online or be hosted locally with docker. It can be used by developers to gather security requirements, design secure code, and verify security tests. Furthermore, it includes features such as:

- an extensivce library code examples
- ASVS and MASVS security checklist
- a platform to train secure coding and hacking skills
- knowledgebase items that are mapped to security items

Furthermore, SKF is supported by companies such as ING, Microsoft and Google.

Table 3.1: User Stories

3.3 Proposed Solution

This thesis proposes SKF to design secure mobile applications. SKF allows security to be integrated into the SDLC for specific sprints. However, it is not an appropriate solution for security risk assessment or threat modeling. Therefore, this thesis concerns only with generating security requirements through SKF and how to work with those set of requirements. This section will discuss the funtional requirements of CheFeed, and what assets needs to be secured and their security requirements generated by SKF.

3.3.1 User Stories

The user stories in Table 3.1 describes the functional requirements for CheFeed.

3.3.2 Security Requirements

Users are essential and important assets of CheFeed. Therefore, it is critical that they can securely store their credentials on their phone. As described in Table 3.1, users register with their email and password and need to authenticate with those credentials. Without proper security in place, malicious users would be able to retrieve the end users credentials and wreak havoc on their accounts.

As seen in the latest installment of the Top 10 Mobile Risk described in Table 3.1, "Insecure Authentication" ranks number four in the list in the latest OWASP Top Ten, "Broken Access Control" ranks as the top security risk in web applications. The security requirements for CheFeed are all for the type mobile applications using maturity level 1. The requirements form the foundation to design and implement the security features into CheFeed.

Table 3.2: Users and Authentication Sprint Configurations

Category	**Configurations**	**Option**
Authentication and Session Management Requirements	Abnormal account activity controls.	*No*
	Authentication/Authorization requirements and lifecyle controls	*Yes*
	Session management requirements and lifecyle	*Yes*
Data Storage and Privacy Requirements	Does your application keep sensitive data and share it with others?	*Yes*
	Does your Application keep sensitive data on the client side?	*Yes*
	Does your application need to provide privacy angainst prying eyes?	*Yes*
	Does your application process personal identifiable information?	*Yes*
Cryptography Requirements	Does your application need to use symmetric encryption	Yes
	Security best practice requirement	*Yes*
	Does your application need to generate secure random numbers for encryption	Yes

Users and Authentication

This subsection describes the security requirements as they are generated by SKFs security expert system for the user and authentication sprint of CheFeeds SDLC. The configuration for this sprint is described in Table 3.2. To ensure the best security requirements for CheFeeds user authentication and session management, security controls from three different categories from the MASVS we're gathered. A summary of the generated requirements can be seen in 3.3. In total CheFeed has a total of thirteens security requirements based on the sprint configurations for this sprint.

Table 3.3: Users and Authentication Sprint Configuration Summary

	Description
2.1	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.
2.2	No sensitive data should be stored outside the app container or system credential storage facilities.
2.3.	No sensitive data is written to application logs.
2.5.	The keyboard cache is disabled on text inputs that process sensitive data.
2.6	No sensitive data is exposed via IPC mechanisms
2.7	No sensitive data, such as passwords or pins, is exposed through the user interface.
4.1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint.
4.2	If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials.
4.3	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.
4.4	The remote endpoint terminates the existing session when the user logs out.
4.5	A password policy exists and is enforced at the remote endpoint.
4.7	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.
4.12	Authorization models should be defined and enforced at the remote endpoint.

Chapter 4

Security Solution Design

This chapter will discuss the solution design to develop CheFeed with a security first approach. First the overall software architecture is described followed by the requirements' specification through user stories and acceptance criteria.

4.1 Software Architecture

4.1.1 Authentication Flow

4.1.2 Authorization Flow

4.2 Requirement Specification

In this section the requirements are described in the form of user stories and their correlated acceptance criteria. The acceptance criteria will be generated by SKF and validate the user stories. Table 4.1 provides an overview of all features for CheFeed and which ones require security verifications.

4.2.1 Gathering Security Controls

For CheFeed it is important that users can securely create accounts and authenticate with their credentials. Furthermore, user authorization must be configured

Table 4.1: CheFeed feature descriptions

	Epic	**Description**	**Require Security Verification**
1	Authentication & Authorization	Create new user account	Yes
2	Authentication & Authorization	Login with created user credential	Yes
3	Authentication & Authorizaation	Logout	Yes
4	Recipes	View all recipes	No
5	Recipes	View single recipe	No
6	Recipes	Create recipe	Yes
7	Recipes	Delete recipe	Yes
8	Recipes	Update recipe	Yes
9	Reviews & Recipes	Create recipe review	Yes

properly. Therefore, the feature development is configured with verifications from two categories, “Data Storage and Privacy Requirements” and “Authentication and Session Management Requirements”. The summary of the sprint configuration is described in Table 3.1. In total fourteen verifications items were generated. Although verification item **4.2** can be ignored, since CheFeed implements a stateless token-based authentication mechanism. As a result, twelve verifications items can be mapped as acceptance criteria to the proper user story.

4.2.2 User Authentication and Session Management

User authentication is an essential part for CheFeed. It allows users to add new recipes to the app. Therefore, users must be able to create a user account first. Table 4.3 describes the user story and acceptance criteria for this feature. As is described, a password policy is enforced at the remote endpoint. Table 4.4 describes how users can login and what security verifications needs to be put in place. Finally, users need to be able to logout as described int Table 4.5.

Table 4.2: Authentication and authorization sprint summary

ID	Description
2.1	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys
2.2	No sensitive data should be stored outside of the app container or system credential storage facilities
2.3	No sensitive data is written to application logs
2.5	The keyboard cache is disabled on text inputs that process sensitive data
2.6	No sensitive data is exposed via IPC mechanisms
4.1	If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint
4.2	If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials
4.3	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm
4.4	The remote endpoint terminates the existing session when the user logs out
4.5	A password policy exists and is enforced at the remote endpoint
4.7	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire
4.12	Authorization models should be defined and enforced at the remote endpoint

Table 4.3: User registration

Description	*As a* user *I can* register with a name, email and password
Acceptance Criteria	• Given that a password policy is in place at the remote endpoint which ensures that the password is at least 8 characters long, has a maximum length of 64 characters, is not silently truncated, has no character limitation • No sensitive data is exposed through the user interface • Keyboard cache is disabled for password input • Password is not written to application logs

Table 4.4: User login

Description	*As a* user I can login with my registered email and password
Acceptance Criteria	• No sensitive data is written to application logs • System credential storage facilities needs to be used to store sensitive data such as the authorization token • The keyboard cache is disabled on text inputs • No sensitive data is exposed via IPC mechanesims • Authentication is performed at the remote endpoint • The server provides a token that has been signed using a secure algorithm

Table 4.5: User logout

Description	*As a* user I can logout
Acceptance Criteria	• The remote endpoint terminates the existing session when the user logs out • Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire

Chapter 5

Testing and Implementation

Chapter 5 discusses the security tests performed against the solution design covered in Chapter 4. The security tests are guided by the MSTG.

5.1 Verifying Secure Data Storage

5.2 Verifying Secure Authentication and User Management

Chapter 6

Evaluation

This chapter will evaluate the process of developing mobile applications with React Native and implementing security requirements based on SKF.

6.1 Mobile Application Development

React Native has allowed for a relative easy and fast development time. Furthermore, it is easy to learn when the developer is already familiar with JavaScript and React DOM. Although the introduction of JavaScript on mobile platforms may cause security concern, the development might outweigh those concerns. However, if a proper security risk assessment and threat modeling is done, a development team can gather proper security requirements and still deliver secure mobile application no matter the programming language or the technology stack being used. In addition, security standards such as the ASVS and MASVS provided by OWASP ensures that users should do proper risk assessment or threat modeling. However, being the frontend developer and security requirement engineer has proven to be difficult and time-consuming. Though gathering the security requirements is relatively easy due to SKFs security expert system, also being the frontend developer is a difficult task. In practice, the security engineer or security champion should delegate the development to the software developers.

6.2 Designing Secure Applications

OWASP is a great source for security tools. They have provided great security standards such as the ASVS and MASVS, as well as security testing guides such as the MSTG. SKF benefits from these indivual OWASP project and provides an easy-to-use web application to access and generate security requirements that can be configured for each development sprint during the SDLC. Though the ASVS and MASVS are great documents, they are designed such that an organization is free in the usage of their standards. SKF makes it easier to use those standards.

Chapter 7

Conclusion and Recommendation

The final chapter discusses the conclusion of the thesis and any further recommendation based on the findings.

7.1 Conclusion

A security first approach to software development is important in today's modern world. However, not all application are processing the same types of assets. A banking applications might require a lot more security requirements than a simple to do app. Therefore, software developers need to be properly trained to understand what security risks their application might face and how to mitigate against different attacks. Fortunately security standards such as the ASVS and MASVS together with security testing guides such as the MSTG exist to assist developers in developing secure applications by design. SKF allows developers to easily gather security requirements from both the ASVS and MASVS and configure them for each feature sprint during the SDLC. In addition, SKF provides links to the MSTG to quickly access information on security testing for mobile applications. Furthermore, SKF includes features such as security labs and courses to train teams and individual developers who need or want to level up their security knowledge. Organization and individual developers would benefit from integrating the framework into their SDLC.

7.2 Recommendation

SKF is a great tool to easily gather security requirements for development projects. However, it can not be used to perform a security risk assessment or threat modeling. Therefore, it is recommended to combine the framework with projects such as OWASP SAMM or the Microsoft SDL. Both introduce security throughout all phases of the development process, where SKF can provide guidance in the training, requirements, design, implementation and verification phase. A combination of such security assurance processes together with SKF can therefore increase the security of any given application, either be that a web or mobile application.

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] 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.

[3] *About the OWASP Foundation*. URL: https://owasp.org/about (visited on 04/02/2022).

[4] 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.

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

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

[7] 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.

[8] 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.

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

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

[11] Wilhelm Hasselbring. "Software architecture: Past, present, future". In: *The Essence of Software Engineering*. Springer, Cham, 2018, pp. 169–184.

[12] *What is a Container?* URL: `https://www.docker.com/resources/what-container/`.

[13] Karl Matthias. *Docker : up and running*. Sebastopol, CA: O'Reilly, 2015. ISBN: 9781491917572.

[14] Arthur Langer. *Guide to software development : designing and managing the life cycle*. London New York: Springer, 2012. ISBN: 9781447167990.

[15] Shylesh S. "A Study of Software Development Life Cycle Process Models". In: *SSRN Electronic Journal* (2017). DOI: `10.2139/ssrn.2988291`. URL: `https://doi.org/10.2139%2Fssrn.2988291`.

[16] Unnati S. Shah, Devesh C. Jinwala, and Sankita J. Patel. "An Excursion to Software Development Life Cycle Models". In: *ACM SIGSOFT Software Engineering Notes* 41.1 (Feb. 2016), pp. 1–6. DOI: `10.1145/2853073.2853080`. URL: `https://doi.org/10.1145%2F2853073.2853080`.

[17] 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`.

[18] Martin Fowler, Jim Highsmith, et al. “The agile manifesto”. In: *Software development* 9.8 (2001), pp. 28–35.

[19] James Helfrich. *Security for software engineers.* Boca Raton: CRC Press, 2019. ISBN: 9781138583825.

[20] B. Potter and G. McGraw. “Software security testing”. In: *IEEE Security amp Privacy Magazine* 2.5 (Sept. 2004), pp. 81–85. DOI: `10.1109/msp.2004.84`. URL: `https://doi.org/10.1109%2Fmsp.2004.84`.

[21] Matteo Meucci and Andrew Mullur. *OWASP Testing Guide v4.0.* Tech. rep. Open Web Application Security Project, 2014.

[22] Sven Schleier Carlos Holguera Bernhard Muller and Jeroen Willemsen. *OWASP Mobile Application Security Verification Standard v1.4.2.* Tech. rep. Open Web Application Security Project, Jan. 2022.