Designing Secure Mobile Applications with OWASP Security Knowledge Framework

Imanuel Febie
Computer Science
BINUS International University
Jakarta, Indonesia
imanuel.febie@binuc.ac.id

Raymond Bahana, S.T., M.Sc
Computer Science
Binus International University
Jakarta, Indonesia
rbahana@binus.edu

Ardimas Andi Purwita, S.T., M.T., Ph.D.
Computer Science
Binus International University
Jakarta, Indonesia
ardimas.purwita@binus.edu

***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.**

***Index Terms*—Software Security, Software Development, Secure Software Development, Mobile Application Development, Secure Software Development Life Cycle, OWASP, SKF**

I. Introduction

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 development. 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.

A. Aims and Benefits

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.

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.

B. 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".

C. Structure

The thesis is divided into seven chapters each building upon the previous ones. In chapter 1 the background of the study is introduced together with the study's aims and benefits and the authors' scope.

Chapter 2 familiarizes the reader with fundamental theories and framework that are essential to this study. The include topics such as information security, software engineering and the SDLC. Lastly, it discusses well established security awareness reports and security standards such as the ASVS and MASVS.

Chapter 3 provides an introduction and analysis to SKF, the main topic of this study. Furthermore, in chapter 4 and 5 the solution design and solution implementation is presented based on the security requirements and security design with SKF.

Chapter 6 discusses the results of the study. Lastly, in chapter 7 the authors' recommendation and conclusion is presented.

II. Theoretical Foundation

A. 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.

1) The Confidentiality, Integrity, and Availability Triad

Security can be complicated as discussed in Section **??**. Nonetheless, as described in Table 1, the CIA triad, provides a model to think about and discuss security concepts. It is

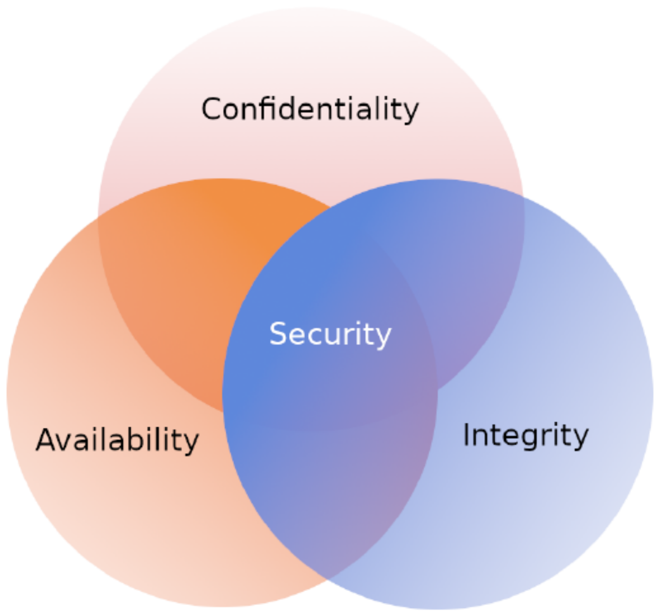


Fig. 1. The CIA Triad

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].

a) Confidentiality

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

b) 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.

c) Availability

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

2) The Gold Standard

a) 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.

b) 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.

c) 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.

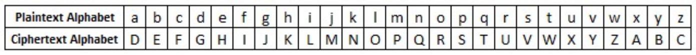
d) Auditing

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.

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



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

Fig. 2. Ceasar Cipher

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. After the fall of the Roman Empire, cryptology was flourishing in the Arabic world [16].

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.

B. *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].

C. *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.

D. *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 i0.5nclude Message-Digest 5 (MD5), MD2, MD4, SHA-2, and RACE.

a) 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.

E. *Software Engineering*

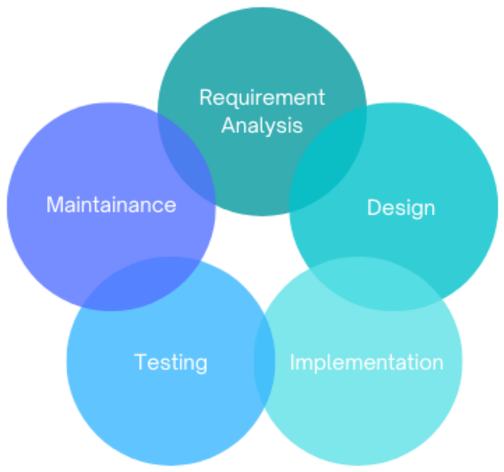
Software is embedded into today's society, and it ranges from simple embedded systems to complex, worldwide information systems. There is no natural limit to the potential of software. However, because of this freedom, software can easily become highly complex, difficult to comprehend, and costly to 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.

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

Fig. 3. The SDLC phases



and so forth led to a countermovement 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 3.

F. *Software Security Awareness and Security Standards*

Today's software demands the highest security standard from companies and developers. Although many security standards exist, this thesis will cover the ones most relevant to the study. These include standards such as the National Institute of Standards and Technology (NIST),lists such as the Common Weakness Enumeration (CWE) and projects from OWASP such as the OWASP Top 10, OWASP ASVS, and the OWASP MASVS.

1) The Open Web Application Security Project

OWASP is a leader in software security. As a non-profit organization focusing on the improvement of software security, it has become a source for security tools, resources, education and training. Furthermore, it has a worldwide community of software security professionals. Security awareness documents such as the OWASP Top 10 and standards such as the ASVS and MASVS are part of OWASP as well is SKF.

2) The Top 10

The OWASP Top 10 is a regularly updated security awareness document concerning web applications, focusing on the ten most critical risks. Each security risk has a list of mapped CWEs. Table **??** describes the top ten security risks by OWASP Top 10. Although the document exist to bring awareness, it has

TABLE I
THE OWASP TOP 10 AS OF 2021

ID	**Description**
AO1:2021	Broken Access Control
AO2:2021	Cryptographic Failures
AO3:2021	Injection
AO4:2021	Insecure Design
AO5:2021	Security Misconfiguration
AO6:2021	Vulnerable and Outdated Components
AO7:2021	Identification and Authentication Failures
AO8:2021	Software and Data Integrity Failures
A09:2021	Security and Logging and Monitoring Failures
A10:2021	Server-side Request Forgery (SSRF)

not stopped companies to use it as a standard to secure software. The OWASP Top 10 provides some recommendations on how to use it as a standard. Additionally, it provides ways to prevent the common attacks. However, it is still encourages to use a real standard such as the ASVS.

a) Broken Access Control

Implementing a reliable access control mechanism is often underestimated by developers, causing unauthorized access to application content and functions.

b) Cryptographic Failures

Data such as passwords, credit cards numbers, health records, personal information, and business secrets require extra protection. Therefore, it is important to determine how much protection data needs in transit and at rest. When encryption is implemented poorly, attackers can retrieve data in its unencrypted state.

c) Injection

Injection attacks include SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injections. Injection vulnerabilites are introduces when:

- User entered data is not validated, filtered or sanitized
- Dynamic queries or non parametemized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within ORM search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.

d) Insecure Design

Business often fail to determine what level of security design is required, due to a lack of business risk profiling to the software being developed. As a result, software is insecure by design. It is therefore important to include security into the SDLC and ensure security activities are performed from the beginning.

e) Security Misconfiguration

Security misconfigurations are security controls that are configured improperly of left insecure, putting assets at risk. These include configuration such as default accounts and password that are not changed, unnecessary features that are enabled, outdated software and many more. Without proper security in place, systems are at more risk.

f) Vulnerable and Outdated Components

Software often rely on third party components on both the server-side and client-side. Unfortunately, these components may have software vulnerabilities when they are developed at high speed without doing any security tests before publishing.

g) Identification and Authentication Failures

Vulnerabilities in authentication schemes can lead to serious and damaging data breaches. Therefore, developers need to ensure that the user's identity, authentication, and session management is properly confirmed.

h) Software and Data Integrity Failures

Moderns software architecture has become more and more complex. As a result applications often rely on plugins, libraries, or modules from untrusted sources, repositories and content delivery networks. Critical data and software updates are often added to the delivery pipeline without verifying their integrity resulting and software data integrity failures [20].

i) Security Logging and Monitoring Failures

The ninth category in the OWASP Top 10 cover security logging and monitoring. Loggin and monitoring provides raw data that assists in identifying possible security threats. This category is difficult to test for since it often involves interviewing or asking if attacks were found during a penetration test. The category exist because without logging and monitoring, breaches simply cannot be detected.

j) Server Side Request Forgery

Server Side Request Forgery (SSRF) happen when an attacker can abuse the functionality of the server to read or update internal resources.

3) The Application Security Verification Standard

The ASVS is a security standard that provides developers a list of security requirements for secure development. The standard can be used to establish a level of confidence in security of web applications. The requirements are categorized in 14 categories that are described in table II. Each ASVS category has its objective and set of security requirements. Furthermore, the ASVS divides the risk severity into three levels. It is up to the organization or development team to decide what security requirements must be implemented. This can be done in the early phases of the SDLC during planning and the requirements' analysis phase.

a) ASVS Level 1

Applications achieve Level 1 if it sufficiently defends against security vulnerabilities that are easy to discover, and are included in the OWASP Top 10. Level 1 is the bare minimum that all applications should strive when implementing security. Furthermore, Level 1 security requirements can be tested against automatically by tools.

b) ASVS Level 2

Applications achieve Level 2 - or standard - if it sufficiently defends against most the of security risks that are associated with today's software. It guarantees that security controls are in place, effective, and used within the application. Level 2 applications include applications that handle significant business-to-business transactions, such as healthcare information, implement critical business or sensitive functions, or other sensitive assets.

TABLE II
ASVS CATEGORIES

ID	Category
V1	Architecture, Design, and Threat Modeling
V2	Authentication
V3	Session Management
V4	Access Controll
V5	Validation, Sanitization and Encoding
V6	Stored Cryptography
V7	Error Handling and Logging
V8	Data Protection
V9	Communication
V10	Malicious Code
V11	Business Logic
V12	Files and Resources
V13	API and Web Service
V14	Configuration

c) ASVS Level 3

The highest verification level is the ASVS Level 3. Level 3 applications are commonly reserved for application that require a substantial levels of security verification, such as applications that are developed for the military, health and safety, critical infrastructure, and so on.

4) The Mobile Application Security Verification Standard

The MASVS is a security standard similar to the ASVS, but offers a security standard for mobile applications specifically. Closely related to the MASVS is the MSTG, which offers test cases against the security requirements defined in the MASVS. Like the ASVS, the security requirements defined in the MASVS are devided into different categories that are described in table III. As seen, the MASVS has lesser categories compared with the ASVS, and also focuses a lot more on how mobile applications handle, store and protect sensitive data. As with the ASVS, which security requirements to implement needs to be decided during the planning and requirements analysis phase of the SDLC by the team responsible for the development of the application.

The MASVS defines two verification levels, MASVS-L1 and MASVS-L2. Additionally, it adds a set of *reverse engineering resiliency requirements*, MASVS-R.

a) MASVS Level 1: Standard Security

The MASVS-L1 is the standard security level all mobile applications should adhere to. Implementing MASVS-L1 ensures that basic security requirements in terms of code quality, handling of sensitive data, and interaction with the mobile environment is fulfilled. Testing must be done to verify wether the security controls are implemented correctly.

TABLE III
MASVS CATEGORIES

ID	Category
V1	Architecture, Design and Threat Modeling
V2	Data Storage and Privacy Requirements
V3	Cryptography Requirements
V4	Authentication and Session Management
V5	Network Communication Requirements
V6	Platform Interaction Requirements
V7	Code Quality and Build Settings Requirements
V8	Resilience Requirements

b) MASVS Level 2: Defense-in-Depth

Going beyond the standard security requirements introduces mobile applications to MASVS-L2. A threat model has be in place to achieve MASVS-L2. Mobile apps such as banking apps have to adhere to MASVS-L2.

c) Resiliency Against Reverse Engineering and Tampering

The MASVS-R protects mobile applications against client side attacks such as tampering, modding, or reverse engineering to extract sensitive code or data. The MASVS-R may be applied to applications handle highly sensitive data to protect its intellectual property or tamper-proofing an app.

III. Analysis on Security Knowledge Framework

IV. Project History

Experience has taught the SKF project leaders that the state of application security is not adequate to ensure security. Primarily due to the lack of knowledge on secure programming [21]. As a result, SKF intended to solve this issue by developing a guide system in the form of an open source web application for developers to support them in developing secure software by design.

SKF has achieved this by adding security standards such as the ASVS and MASVS into their web application project management feature. Furthermore, contributors have added secure code examples for several popular programming languages and frameworks such as Java, PHP, go, ruby, nodejs, ASP.NET, flask and Django. The web application also has features such as hacking labs and a training platform to train developers, security pentesters and development operations (DevOps).

V. Software Development Project Management

SKF provides many features as discussed previously, this study however will only use SKF to design secure software through its project management feature. Specifically, MASVS security controls will be used to develop secure user authentication and user management throughout the requirements analysis, design, implementation and testing phases of the SDLC.

A. Requirements Analysis

During the requirements' analysis phase the question, "What do we want?" is answered. From a security perspective, it is important to establish what security controls the software feature needs. In SKF, security requirements are defined when starting a new feature development sprint within a project.

B. Design

After establishing what is required for the feature sprint, we seek to answer "How do we get what we want?". Acceptance criteria can be defined based on the requirements established in the previous phase. During the design phase, developers come to understand what potential threats are introduced for the feature that is being developed and how they can be mitigated.

C. Implementation

After having defined what is required and how to get there, the implementation phase can begin. Based on the acceptance criteria code can be written not only focused on the functional requirement, but on the security requirement as well.

D. Testing

To ensure the security controls behave as they should, they need to be tested. MASVS-L1 allows for complete automated testing. Guidance on how to perform tests related to mobile security can be found in the MSTG. Links to the associated test for the security controls

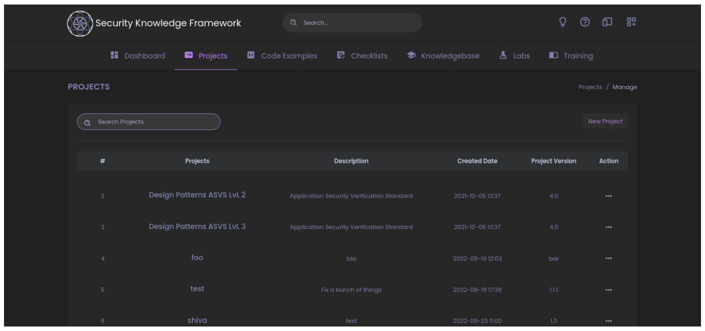
E. Setup and Installation on Linux

SKF is completely free to use and open source. The entire source code is hosted on github. This allows organizations and developers to use SKF however they want, and even modify it to their needs. SKF hosts an online demo of their web application where users can get familiar with the web application. However, in a professional environment it is recommended to set up and install SKF on a personal server. d SKF provides several installation options to host the web application. For this study, SKF was hosted on a local Arch Linux machine through docker. To run SKF on a Linux machine *git*, *docker*, *docker-compose* and *minikube* needs to be installed. On Arch Linux they can be installed throuch *pacman*. Debian based distributions can install the required software with *apt*. If docker is set up properly together with minikube, SKF can be cloned from its official repository on github. After cloning the repository, change directory into `skf-flask` and run `docker-compose up`. If no errors occur, the application can be accessed inside a browser on *localhost* as seen in figure **??**. Step by step the process to install and setup SKF includes:

1) Install *docker, docker-compose*, and *minikube*
2) Clone SKF repository from `https://github.com/blabla1337/skf-flask`
3) Change directory into `skf-flask`
4) Run `docker-compose up` inside the root folder

F. SKF Project Management

When navigating to the project tabs in SKF, the user is shown a list of design pattern projects. Typically, the user wants to create their own project and set of requirements for a specific feature. To achieve this, the user can simply create a new



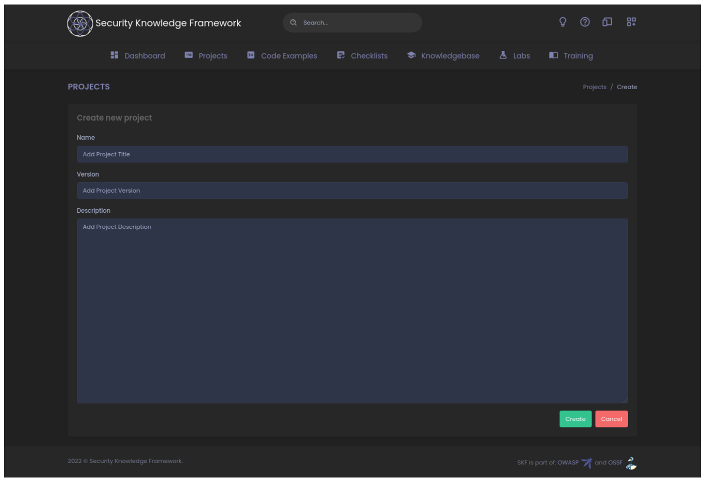
PROJECTS

Projects / Manage

Search Projects

#	Projects	Description	Created Date	Project Version	Action
2	Design Patterns ASVS LvL 2	Application Security Verification Standard	2021-10-05 13:37	4.0	...
3	Design Patterns ASVS LvL 3	Application Security Verification Standard	2021-10-05 13:37	4.0	...
4	foo	bla	2022-08-19 12:03	bar	...
5	test	Fix a bunch of things	2022-08-19 17:38	1.1.1	...
6	shiva	test	2022-08-23 11:00	1.0	...

Fig. 4. SKF Project



PROJECTS

Projects / Create

Create new project

Name: Add Project Title

Version: Add Project Version

Description: Add Project Description

2022 © Security Knowledge Framework.

SKF is part of: OWASP and OSSF

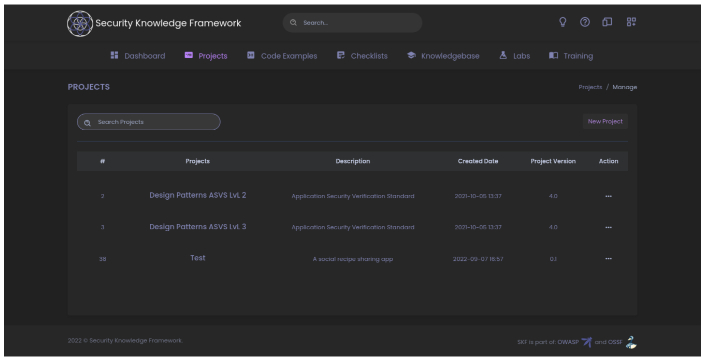
Fig. 5. SKF New Project

project. The button is located in top right as seen in figure 4. This will bring the user to a page where the project details can be entered such the *project name*, *version* and *description* as seen in figure 5. After the project is finished, it is added to the list of projects in the projects' page as seen in figure 6.

When navigating to the created project, the user can create new requirements for a new feature or an existing feature by clicking the **Generate Requirements** button seen in figure 7. When generating new requirements for a feature, users can choose the *checklist type*, *verification level*, *categories* and configure their sprint so that only the security requirements needed are generated. Lastly, the security expert system will ask whether the set of security requirements are to be added to an existing feature or a new one. Chapter 4 will discuss the complete process for the design of CheFeeds authentication and session management feature sprint

VI. Alternative Solutions

As far as the authors' knowledge goes, there are no solutions that are similar to SKF in its entirety. The best alternative solutions are the ASVS and MASVS and the related testing guide series from OWASP. SKF is unique as it provides a complete security framework and resource for organizations and developers who need to implement security. However,



PROJECTS

Projects / Manage

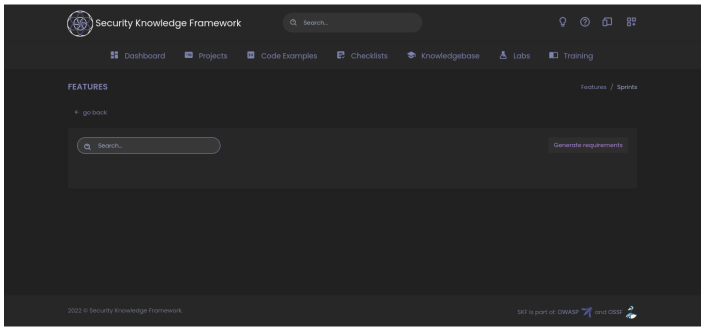
Search Projects

#	Projects	Description	Created Date	Project Version	Action
2	Design Patterns ASVS LvL 2	Application Security Verification Standard	2021-10-05 13:37	4.0	...
3	Design Patterns ASVS LvL 3	Application Security Verification Standard	2021-10-05 13:37	4.0	...
38	Test	A social recipe sharing app	2022-09-07 16:57	0.1	...

2022 © Security Knowledge Framework.

SKF is part of: OWASP and OSSF

Fig. 6. SKF new added project named "test"



FEATURES

Features / Sprints

2022 © Security Knowledge Framework.

SKF is part of: OWASP and OSSF

Fig. 7. SKF project features

processes such as the OWASP Software Assurance Maturity Model (SAMM) or the Microsoft Security Lifecycle (SDL) may act as a complement to SKF. Both ensure that security activities are done throughout the whole SDLC. This section will briefly introduce them.

A. OWASP Software Assurance Maturity Model

The OWASP SAMM model provides a simple to use prescribed model which is fully defined, and measurable. SAMM assists organizations in analyzing the current software security practices, development of a security program in defined iterations, show gradual improvements in secure practices, and define and measure security-related activities. A complete overview of SAMMs model is illustrated in figure 8.

Fig. 8. OWASP SAMM model

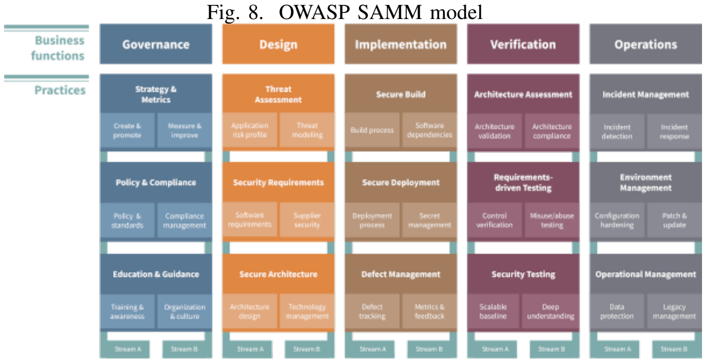


TABLE IV
MICROSOFT SDL PRACTICES

Practice #	**Description**
Practice 1	Provide Training
Practice 2	Define Security Requirements
Practice 3	Define Metrics and Compliance Reporting
Practice 4	Perform Threat Modeling
Practice 5	Establish Design Requirements
Practice 6	Define and Use Cryptographic Standards
Practice 7	Manage the Security Risks of Using Third-Party Components
Practice 8	Use Approved Tools
Practice 9	Performing Static Analysis Security Testing (SAST)
Practice 10	Perform Dynamic Analysis Security Testing (DAST)
Practice 11	Perform Penetration Testing
Practice 12	Establish a Standard Incident Response Process

B. *Microsoft Security lifecycle*

Microsofts SDL assists developers building highly secure software, address security compliance requirements, and reduce development costs. It keeps up to date with current technologies such as the cloud, Internet of Things, and artificial intelligence. The SDL includes a set of practices that support security assurance and compliance requirements. The set of practices included by the SDL are described in table IV.

VII. SOLUTION DESIGN

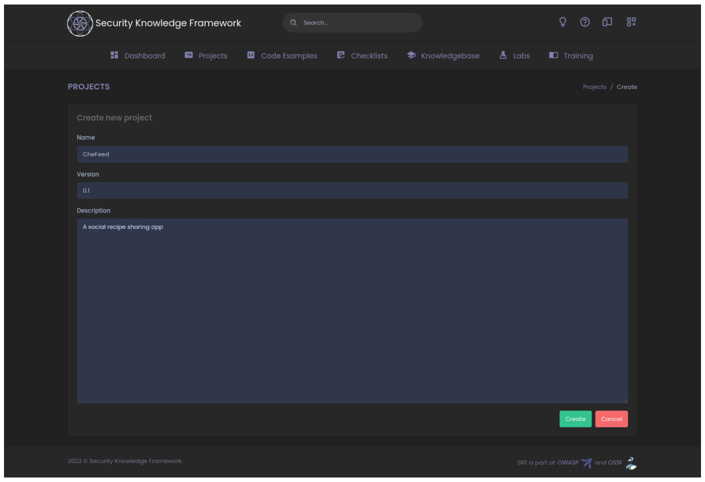
A. *Requirement Analysis*

An important mechanism of many applications is allowing users to login using a combination of their username or email, and password. Authentication gives user certain authorization, such as posting recipes and managing them. As was discussed in II-F2, features such as access control and authentication is still a prevalent security risk. Secure user authentication and authorization therefore is important for users of CheFeed. The feature development for user authentication and session management includes abilities such as *user registration*, *user login*, *user logout*, and the *authorization* to *create, read, update*, and *delete* recipes.

This section will briefly present the security requirement analysis for user authentication and user management that has been acquired by using SKF. First, a new project is created with SKF as seen in figure 9. This will provide the ability to generate requirements for the created project. Appropriately, the project is named CheFeed.

1) Generating Security Requirements

The security requirements for user authentication and session management were configured such that it had had the MASVS-L1 security controls in place. The categories considered for its feature development included:



Create new project

Name: CheFeed

Version: 0.1

Description: A social recipe sharing app

Fig. 9. Create CheFeed Project

- Data Storage and Privacy
- Authentication and Session management

Figure 10, 11, 12, 13, 14, and 15 illustrates the selection process of SKFs security expert system and will be explained briefly.

a) Checklist Type

When generating new security requirements for a feature sprint, SKF allows to generate security requirements for either web applications or mobile applications. Additionally, a custom set of requirements can be generated in the dropdown menu. For CheFeed a list of mobile application security requirements are required. Therefore, the mobile application type is selected in this process as seen in figure 10. This will generate security requirements based from the MASVS.

b) Security Maturity Level

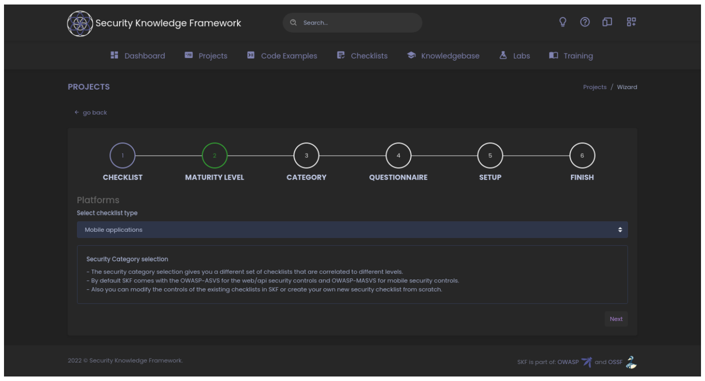
Figure 11 shows the second step in the security expert wizard is to decide what security maturity level the feature requires, with the options Level 1, Level 2, and Level 3. However, the MASVS does not have a MASVS-L3 maturity level. This seems to be a bug that can be ignored for now. For this study CheFeed implements MASVS-L1, which is recommended for all applications and ensures that it follows the bare minimum.

c) Categories

The next step in process is choosing security requirements from the MASVS categories the feature needs as seen. SKF lists out all the MASVS categories and the desired categories can be checked. For the development of CheFeeds user authentication and user session management, requirements from the categories *Data Storage and Privacy* and *Authentication and Session Management* were selected as seen in figure 12. The first category provides security controls to protect sensitive data such as user credentials and private information. The second category ensures that that user accounts and sessions are managed securely on the client side, although most of the logic happens at the remote endpoint.

d) Sprint Configuration

The sprint configuration step displays a great feature of SKFs security expert system in the requirement analysis phase. This



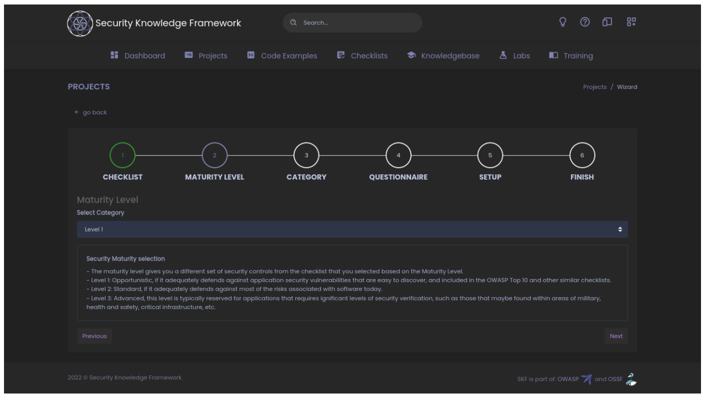
Platforms

Select checklist type: Mobile applications

Security Category selection

- The security category selection gives you a different set of checklists that are correlated to different levels.
- By default SKF comes with the OWASP-ASVS for the web/api security controls and OWASP-MASVS for mobile security controls.
- Also you can modify the controls of the existing checklists in SKF or create your own new security checklist from scratch.

Fig. 10. SKF Security Expert - Select checklist type (security standard)



Maturity Level

Select Category: Level 1

Security Maturity selection

- The maturity level gives you a different set of security controls from the checklist that you selected based on the Maturity Level.
- Level 1: Opportunistic, if it adequately defends against application security vulnerabilities that are easy to discover, and included in the OWASP Top 10 and other similar checklists.
- Level 2: Standard, if it adequately defends against most of the risks associated with software today.
- Level 3: Advanced, this level is typically reserved for applications that requires ignificant levels of security verification, such as those that maybe found within areas of military, health and safety, critical infrastructure, etc.

Fig. 11. SKF Security Expert - Select maturity level

process allows to easily generate the security requirements from the chosen MASVS categories selected in the step before. Based on the answers provided, the correct security requirements will be filtered. Figure 13 shows what configurations were set for CheFeeds user authentication and user sessions feature.

e) Create Sprint

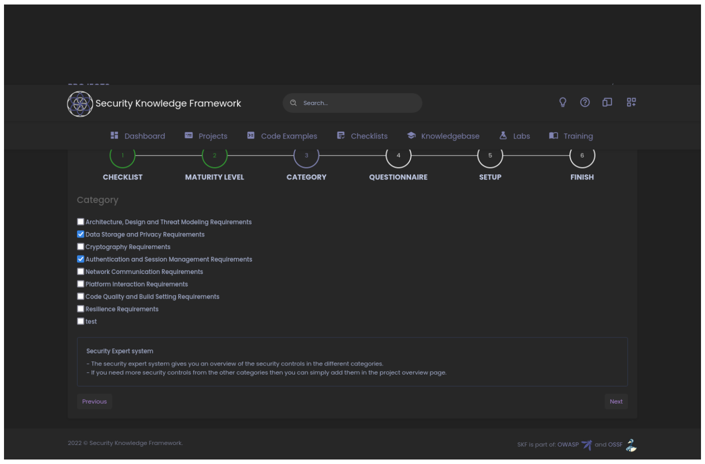
Lastly we can create a new feature for the created project or add to an existing one. Since this is a new project with security requirements for a new feature, a new feature is created as seen in figure 14. A name, version and description can be provided to the feature. Lastly, everything can be submitted to generate the security requirements for the feature as seen in figure 15.

f) Requirements Summary

The process described before generates a summary of security requirements seen in figure 16 that can be used further in the design, implementation and testing phase of the SDLC. Table V gives a more clear overview of the security requirements summary from figure 16. Fourteen security requirements were gathered in total for the development of user authentication and user session management in CheFeed.

B. Design

The requirements set the foundation on how the security of the authentication and session management are to be designed. In this section a detailed description of the security design is



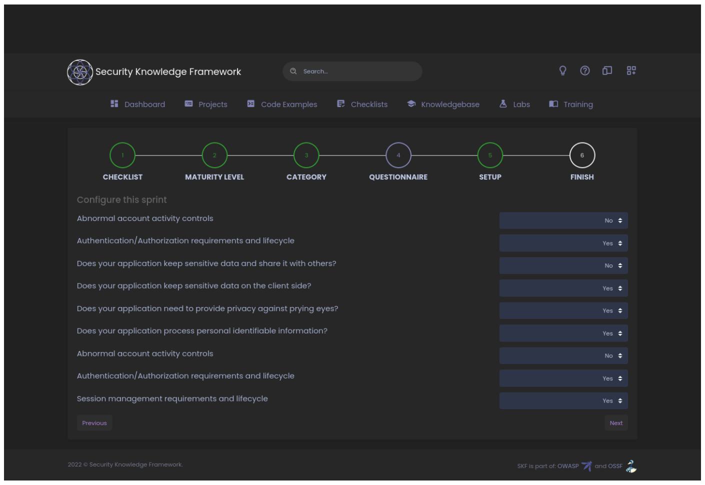
Category

- ☐ Architecture, Design and Threat Modeling Requirements
- ☑ Data Storage and Privacy Requirements
- ☐ Cryptography Requirements
- ☑ Authentication and Session Management Requirements
- ☐ Network Communication Requirements
- ☐ Platform Interaction Requirements
- ☐ Code Quality and Build Setting Requirements
- ☐ Resilience Requirements
- ☐ test

Security Expert system

- The security expert system gives you an overview of the security controls in the different categories.
- If you need more security controls from the other categories then you can simply add them in the project overview page.

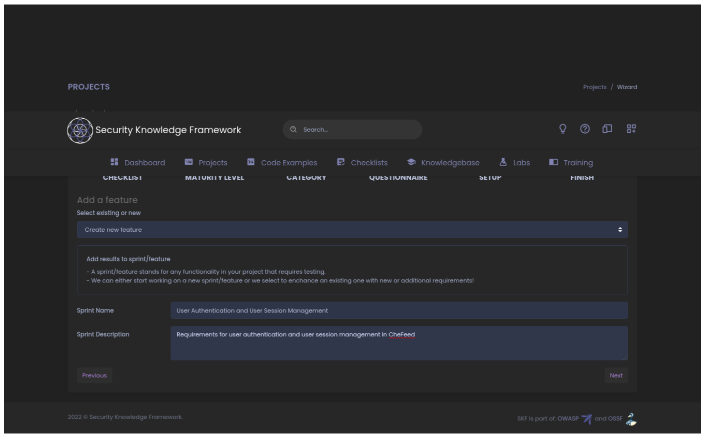
Fig. 12. SKF Security Expert - MASVS Category



Configure this sprint

Abnormal account activity controls	No
Authentication/Authorization requirements and lifecycle	Yes
Does your application keep sensitive data and share it with others?	No
Does your application keep sensitive data on the client side?	Yes
Does your application need to provide privacy against prying eyes?	Yes
Does your application process personal identifiable information?	Yes
Abnormal account activity controls	No
Authentication/Authorization requirements and lifecycle	Yes
Session management requirements and lifecycle	Yes

Fig. 13. SKF Security Expert - Sprint Configuration



Add a feature

Select existing or new: Create new feature

Add results to sprint/feature

- A sprint/feature stands for any functionality in your project that requires testing.
- We can either start working on a new sprint/feature or we select to enchance an existing one with new or additional requirements!

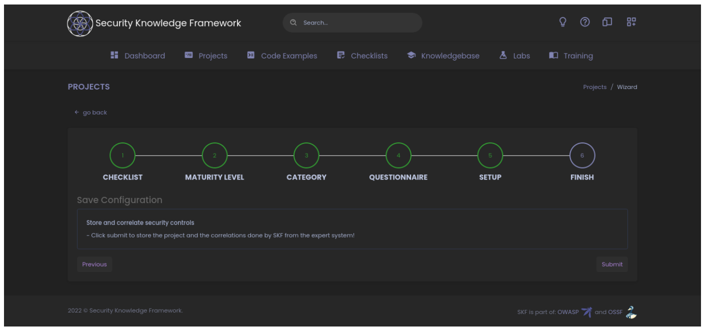
Sprint Name: User Authentication and User Session Management

Sprint Description: Requirements for user authentication and user session management in CheFeed

Fig. 14. SKF Security Expert - Create feature sprint

TABLE V
SUMMARY OF USER AUTHENTICATION AND SESSIONS SECURITY REQUIREMENTS

#	Cate-gory	Description
2.1	Data Storage and Privacy	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys
2.2	Data Storage and Privacy	No sensitive data should be stored outside of the app container or system credential storage facilities
2.3	Data Storage and Privacy	No sensitive data is written to application logs
2.4	Data Storage and Privacy	No sensitive data is shared with third parties unless it is a necessary part of the architecture
2.5	Data Storage and Privacy	The keyboard cache is disabled on text inputs that process sensitive data
2.6	Data Storange and Privacy	No sensitive data is exposed via IPC mechanisms
2.7	Data Storage and Pricacy	No sensitive data, such as passwords or pins, is exposed through the user interface
4.1	Authen-tication and Session Manage-ment	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	Authen-tication and Session Manage-ment	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	Authen-tication and Session Manage-ment	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm
4.4	Authen-tication and Session Manage-ment	The remote endpoint terminates the existing session when the user logs out
4.5	Authen-tication and Session	A password policy exists and is enforced at the remote endpoint

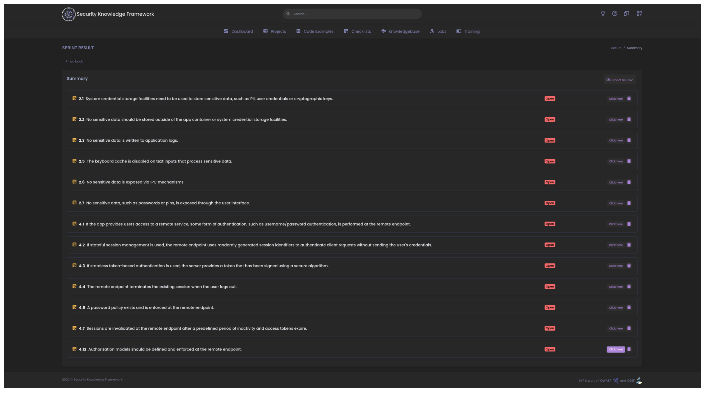


Save Configuration

Store and correlate security controls

- Click submit to store the project and the correlations done by SKF from the expert system!

Fig. 15. SKF Security Expert - Submit



2.1 System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys.	Open	Click here
2.2 No sensitive data should be stored outside of the app container or system credential storage facilities.	Open	Click here
2.3 No sensitive data is written to application logs.	Open	Click here
2.5 The keyboard cache is disabled on text inputs that process sensitive data.	Open	Click here
2.6 No sensitive data is exposed via IPC mechanisms.	Open	Click here
2.7 No sensitive data, such as passwords or pins, is exposed through the user interface.	Open	Click here
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.	Open	Click here
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.	Open	Click here
4.3 If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm.	Open	Click here
4.4 The remote endpoint terminates the existing session when the user logs out.	Open	Click here
4.5 A password policy exists and is enforced at the remote endpoint.	Open	Click here
4.7 Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire.	Open	Click here
4.12 Authorization models should be defined and enforced at the remote endpoint.	Open	Click here

Fig. 16. Summary security requirements

presented. First, the overall system architecture is presented. Furthemore, user stories are presented that describe the functionality, benefits, and security acceptance criteria to properly mitigate security risks. Lastly a sequence diagram is added to each user story.

1) *System Architecture*

CheFeed follows a traditional three-tier software architecture as displayed in figure 17. The client is the mobile application developed with *react native*. A JavaScript library to develop mobile user interfaces for both Android and iOS devices. It sends and receives requests from the API. The API is developed with FastAPI, a python framework to develop fast APIs. Furthermore, two databases are connected to the API. One to store business data and one to temporarily store random user tokens for user authorization whenever a user starts a new user session. All components for CheFeed are containerized with docker except for the client.

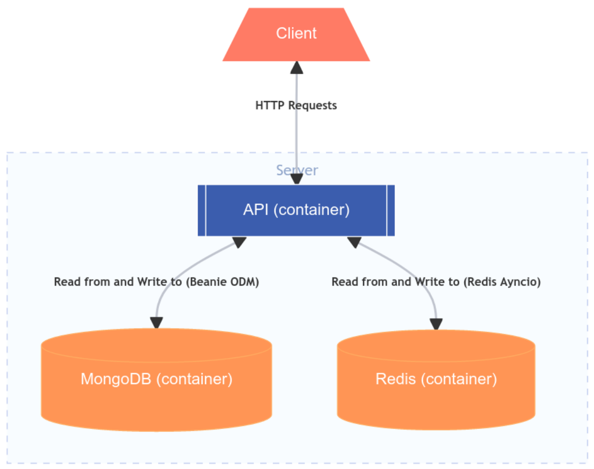
2) *User Stories*

Authentication includes functions such as user registration, login. These are functions that are implemented for CheFeed and have to adhere to the security requirements generated and described in table V.

a) *User Registration*

User registration allows users to authenticate themselves in Chefeed using their account. It is the first functionality for the authentication and user sessions feature development. CheFeeds requires users to provide an email address and password to

Fig. 17. CheFeed system architecture



create an account. Table VI describes the user story for user registration.

b) User Login

User login allows users to authenticate themselves with their registered account. Table VII describes the user story for login and figure 18 illustrates the authentication flow.

VIII. Solution Implementation

A. API Implementation

The API for CheFeed is implemented with FastAPI. A python library to develop APIs with Python 3.6 and above with support for standard Python type hints. FastAPI is chosen for the API implementation due to it being fast and having built-in

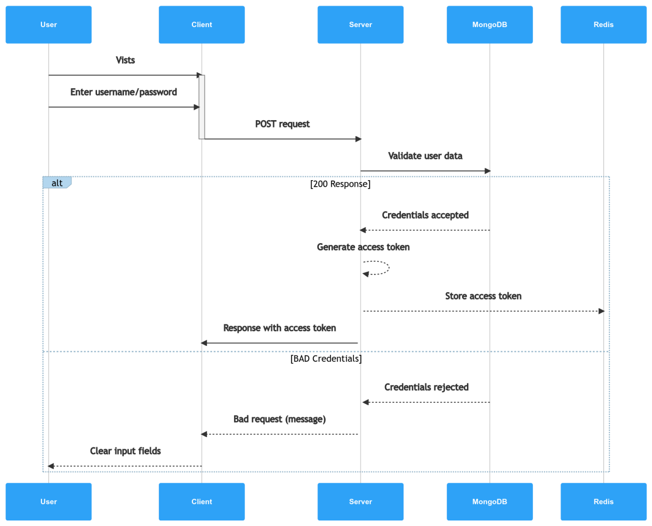


Fig. 18. Authentication flow

TABLE VI
User Story - Register

Description	As a user I want to be able to register an account for the CheFeed app
Benefits	CheFeed hosts user information, including sensitive data. For users to access their stored data, users must be able to authenticate themselves with their registered account. Thus helping ensure that an unauthorized party can access and modify their data
Acceptance Criteria	• Data storage in source code must be analyzed • Ensure all possible functionality in the application are triggered in order to ensure data generation • Check all application generated and modified files and ensure that the storage method is sufficiently secure such as SharedPreferences, SQL databases, Realm databases, Internal Storage, External Storage

TABLE VII
User story - Login

Description	As a registered user I want to be able login into the application
Benefits	CheFeed hosts user data, including sensitive information. Each user has their data stored separately, each with their associated account. To access their stored data, users must be able to authenticate themselves securely.

security features such as authentication. Furthermore, FastAPI also has several external plugins available to enhance and speed up development time. Table VIII describes the implemented security controls for the API.

1) Authentication and Session Management

FastAPI on its own can be used to develop authentication mechanisms. However, the process can be time consuming. Therefore, an external plugin was added to the source code. The plugin *fastapi-users* provides all the features needed to quickly develop user registration, authentication, and options to manage user sessions. Furthermore, it supports both SQL databases and MongoDB.

TABLE VIII
IMPLEMENTED SECURITY CONTROLS

ID	Security control	Done
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	Yes
4.3	If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm	Yes
4.4	The remote endpoint terminates the existing session when the user logs out	Yes
4.5	A password policy exists and is enforced at the remote endpoint	Yes
4.7	Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire	Yes
4.12	Authorization models should be defined and enforced at the remote endpoint	Yes

a) Authentication Backend

Fastapi-users provides methods to manage tokens. The first part is called the *transport* which is responsible for managing how the token is being carried over the request. The second part is called the *transport*, and it is responsible for managing how the token is generated and secured. A combination of the transport and strategy is called the *authentication backend* by fastapi-users.

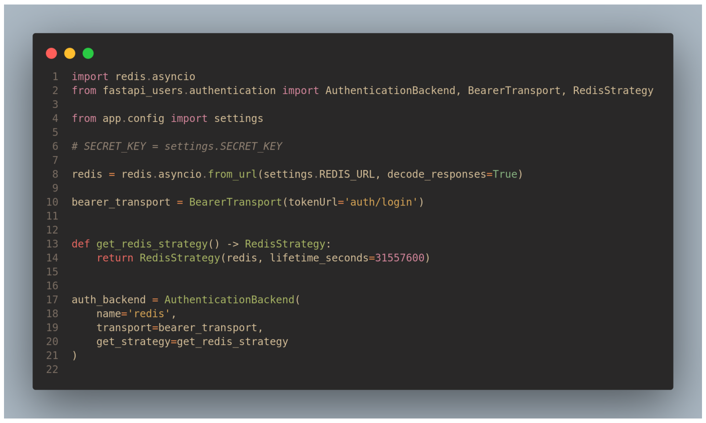
For CheFeed, a *bearer* token transport is implemented together with a separate database to store the token. As seen previously in figure 17 the token is stored in a Redis database. Therefore, user authentication is stateless and token-based. The implementation of the authentication backend using fastapi-users as seen in figure 19, ensures that the security controls *4.3*, *4.4* and *4.12* from table V are in implemented.

Appropriate Authentication

Fastapi-users includes authentication routes to handle user login and user logout without any customization needed. However, the authentication backend and login manager need to be implemented as seen in figure 19 and figure 20. With those implemented, the authentication route can be implemented as seen in figure **??**.

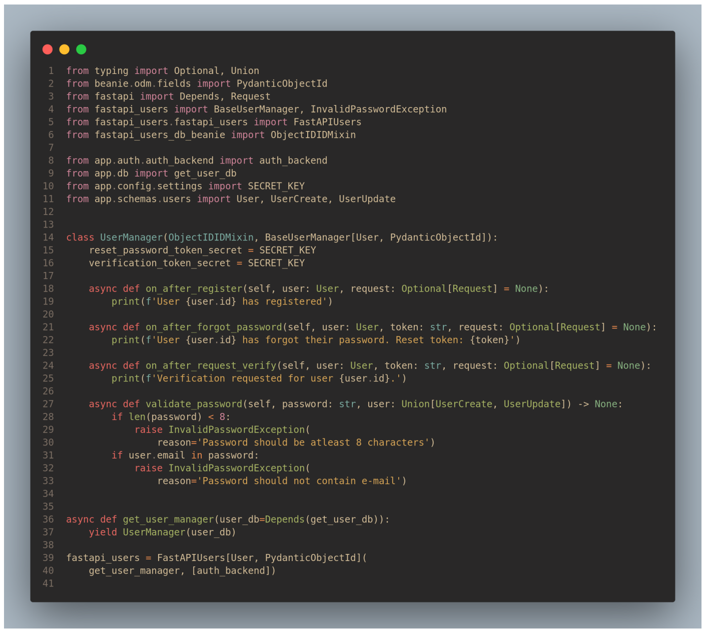
Stateless Authentication

The security control *4.3* states "If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm". The fastapi-users library allows connecting to a Redis database to securely store user tokens that are created upon user authentication and destroyed when logging out. This is handled by the library itself.



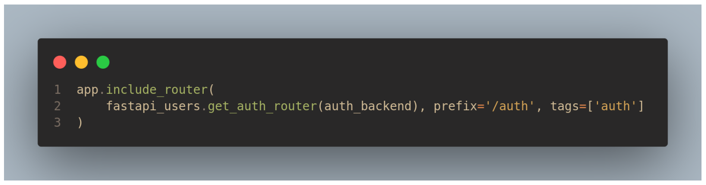
```python
import redis.asyncio
from fastapi_users.authentication import AuthenticationBackend, BearerTransport, RedisStrategy

from app.config import settings

# SECRET_KEY = settings.SECRET_KEY

redis = redis.asyncio.from_url(settings.REDIS_URL, decode_responses=True)

bearer_transport = BearerTransport(tokenUrl='auth/login')


def get_redis_strategy() -> RedisStrategy:
    return RedisStrategy(redis, lifetime_seconds=31557600)


auth_backend = AuthenticationBackend(
    name='redis',
    transport=bearer_transport,
    get_strategy=get_redis_strategy
)
```

Fig. 19. API authentication backend with fastapi-users



```python
from typing import Optional, Union
from beanie.odm.fields import PydanticObjectId
from fastapi import Depends, Request
from fastapi_users import BaseUserManager, InvalidPasswordException
from fastapi_users.fastapi_users import FastAPIUsers
from fastapi_users_db_beanie import ObjectIDIDMixin

from app.auth.auth_backend import auth_backend
from app.db import get_user_db
from app.config.settings import SECRET_KEY
from app.schemas.users import User, UserCreate, UserUpdate


class UserManager(ObjectIDIDMixin, BaseUserManager[User, PydanticObjectId]):
    reset_password_token_secret = SECRET_KEY
    verification_token_secret = SECRET_KEY

    async def on_after_register(self, user: User, request: Optional[Request] = None):
        print(f'User {user.id} has registered')

    async def on_after_forgot_password(self, user: User, token: str, request: Optional[Request] = None):
        print(f'User {user.id} has forgot their password. Reset token: {token}')

    async def on_after_request_verify(self, user: User, token: str, request: Optional[Request] = None):
        print(f'Verification requested for user {user.id}.')

    async def validate_password(self, password: str, user: Union[UserCreate, UserUpdate]) -> None:
        if len(password) < 8:
            raise InvalidPasswordException(
                reason='Password should be atleast 8 characters')
        if user.email in password:
            raise InvalidPasswordException(
                reason='Password should not contain e-mail')


async def get_user_manager(user_db=Depends(get_user_db)):
    yield UserManager(user_db)

fastapi_users = FastAPIUsers[User, PydanticObjectId](
    get_user_manager, [auth_backend])
```

Fig. 20. API login manager with fastapi-users



```python
app.include_router(
    fastapi_users.get_auth_router(auth_backend), prefix='/auth', tags=['auth']
)
```

Fig. 21. API auth router with fastapi-users

TABLE IX
IMPLEMENTED SECURITY CONTROLS ON CLIENT SIDE

ID	Security Control	Done
2.1	System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys	Yes
2.2	No sensitive data should be stored outside of the app container or system credential storage facilities	No
2.3	No sensitive data is written to application logs	Yes
2.5	The keyboard cache is disabled on text inputs that process sensitive data	No
2.6	No sensitive data is exposed via IPC mechanisms	No
2.7	No sensitive data, such as passwords or pins, is exposed through the user interface	No

Authorization Model

The created token also puts in place a mechanism that allows or denies the user access to an endpoint. As a result security control *4.12* from table V can also be marked as done and implemented.

Session Timeout

Lastly, the token lifetime is set. That way sessions are invalidated at the remote endpoint after a predefined period of inactivity as required by security control *4.7*.

B. Client-side Implementation

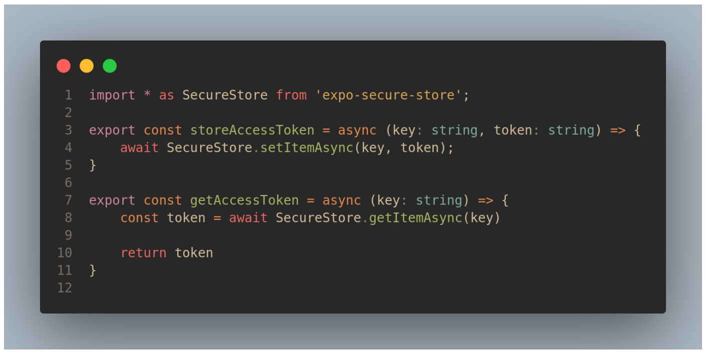
React Native together with the Expo SDK provides means to implement security. In this section the security implementation is presented for the client side. As seen in table IX not all security controls have been implemented unlike the API security controls.

1) Authentication and Session Management

The client-side primarily concerns about the user interface and the user experience. However, user authentication and user access control has to be implemented as well. For example, during the requirements gathering phase it has already been established that tokens that are generated by the remote endpoint need to be stored on the client side in order to provide a better user experience and permitting access to certain content and functions of CheFeed.

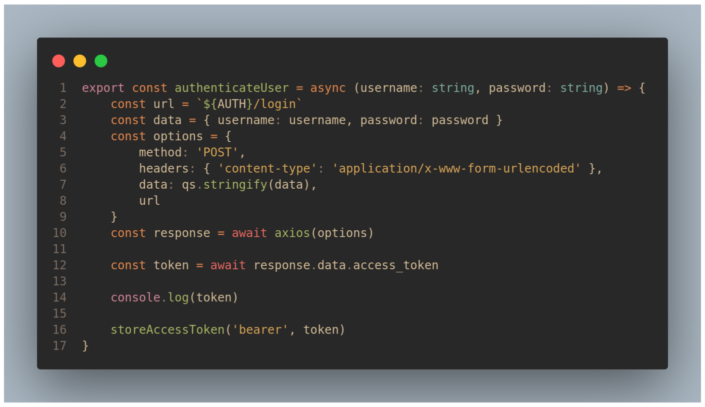
a) Local Storage for Sensitive Data

The only sensitive data CheFeeds requires storing locally is the user authorization token. Besides the benefit of the user not needing to log in every time the user wants to use the app, it also provides the user authorization to certain functions within CheFeed such posting recipes. To achieve secure local storage CheFeeds implements the *SecureStorage* API provided by Expo. The API allows data encryption and decryption using the local storage API from both Android and iOS devices.



```
import * as SecureStore from 'expo-secure-store';

export const storeAccessToken = async (key: string, token: string) => {
    await SecureStore.setItemAsync(key, token);
}

export const getAccessToken = async (key: string) => {
    const token = await SecureStore.getItemAsync(key)

    return token
}

```

Fig. 22. Client secure storage



```
1  export const authenticateUser = async (username: string, password: string) => {
2      const url = `${AUTH}/login`
3      const data = { username: username, password: password }
4      const options = {
5          method: 'POST',
6          headers: { 'content-type': 'application/x-www-form-urlencoded' },
7          data: qs.stringify(data),
8          url
9      }
10     const response = await axios(options)
11
12     const token = await response.data.access_token
13
14     console.log(token)
15
16     storeAccessToken('bearer', token)
17 }
```

Fig. 23. Client authentication

The implementation is seen in figure 22. In figure 23 is shown how the `storeAccessToken()` function is used in the `authenticateUser()` function on line 16. Here it is ensured that whenever the user is authenticated, the token received by the remote endpoint is stored securely.

IX. DISCUSSION

A. Discussion

SKF has proven to be partially successful in the secure SDLC of developing CheFeed. It has shown that it can generate security requirements based on the projects needs. Furthermore, based on the requirements we come to understand what potential security risks are introduced for the development of the user authentication and user session management feature which is useful during the design phase of the SDLC.

Design however does not always directly translate into code implementation. This can be due to a lack of understanding of the programming language or the level of security training the developer has received. Not all security requirements were implemented by the developing team. Furthermore, a lack of security training and understanding of implementing security tests made the testing phase difficult. Even with the security testing guide and mappings provided by SKF, without proper education and training it is still difficult to understand how static analysis tools are to be used for security specifically.

TABLE X
RESULTS IMPLEMENTED SECURITY CONTROLS

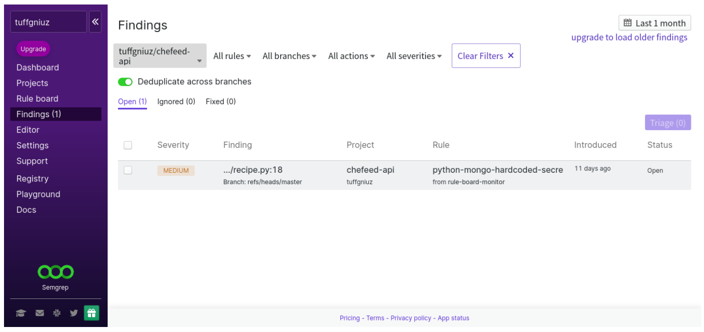
Security Control	**Implemented**
System credential storage facilities need to be used to store sensitive data, such as PII, user credentials or cryptographic keys	Yes
No sensitive data should be stored outside of the app container or system credential storage facilities	No
No sensitive data is written to application logs	Yes
The keyboard cache is disabled on text inputs that process sensitive data	No
No sensitive data, such as passwords or pins, is exposed through the user interface	Yes
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	Yes
If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm	Yes
The remote endpoint terminates the existing session when the user logs out	Yes
A password policy exists and is enforced at the remote endpoint	Yes
Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire	Yes
Authorization models should be defined and enforced at the remote endpoint	Yes

Although the MSTG and its test cases provide detailed explanation, they are technical and do require further research into the recommended static analysis tools and dynamic analysis tools. Therefore, this study mainly resolved to manual code reviews. Table X describes what security controls have been implemented during development of CheFeed for both the API and client-side.

B. Test Results

The static analysis tools on both the API repository and client side repository were added to continuous integration pipeline. Figure **??** shows the test results for the API. The figure shows that for the API one medium severity finding was one found. Here it concerns a security misconfiguration where the secret key is hard coded in the source code.

Although the client side repository had semgrep added to the continous integrated pipeline, it had difficulties in finalizing the scanning progress as seen in figure **??**.



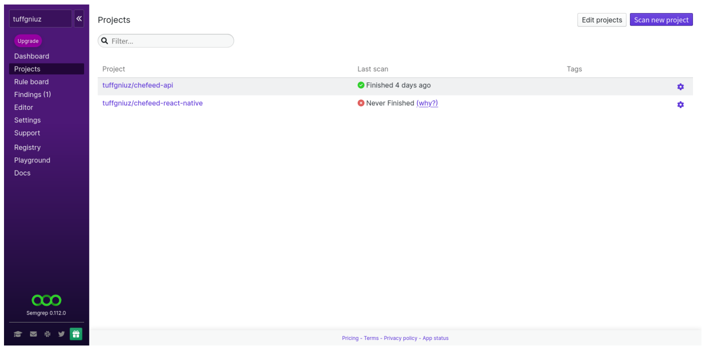
Findings

Deduplicate across branches

Open (1) Ignored (0) Fixed (0)

Severity	Finding	Project	Rule	Introduced	Status
MEDIUM	.../recipe.py:18 Branch: refs/heads/master	chefeed-api tuffgniuz	python-mongo-hardcoded-secre from rule-board-monitor	11 days ago	Open

Fig. 24. SEMGREP results for the API



Projects

Project	Last scan	Tags
tuffgniuz/chefeed-api	Finished 4 days ago	
tuffgniuz/chefeed-react-native	Never Finished (why?)	

Fig. 25. Failed scan on the client side code

X. CONCLUSION AND RECOMMENDATION

A. Conclusion

The study set out to implement SKF in an agile SDLC. As a case study, the development of a full stack application was conducted. This included the development of the API and the clientside. To ensure security activities were performed during the SDLC, SKF was used to do security requirements, security design, secure code implementation, and security testing. In this study SKF was used to achieve MASVS-L1 wich is the bare minimum that all mobile applications should adhere too. Specifically, security requirements were gathered for the development of the authentication and user session management feature of CheFeed. Chapter 5 showed the code implementation based on the security requirements and design from chapter 4. Lastly, security tests were performed in the form of manual tests and static analysis tools. SKF is great in the functional part of the SDLC. However, without a proper and clear process ahead SKF by itself may not suffice.

B. Recommendation

SKF should be used in complement with processes such as OWASP SAMM or Microsoft SDL. Both provide clear security activities throughout the whole SDLC. An important part that was missing during this study, was introducing security training during the SDLC. Without the proper security training, security is difficult to understand and even more difficult to implement. Fortunately, a training platform is provided by SKF with free courses. Future studies might benefit by developing a secure SDLC where SKF is combined with OWASP SAMM or Microsoft SDL.

REFERENCES

[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: 9780133943030,1292096136,9781292096131. URL: http://gen.lib.rus.ec/book/index.php?md5=EB8FF24BBA604674DD15D3E029CA6AB2.

[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.

[20] Sudip Sengupta. *A08:2022 - Software and Data Integrity Failures - Explained*. 2022. URL: https://crashtest-security.com/owasp-software-data-integrity-failures/.

[21] *SKF Introduction*. URL: https://skf.readme.io/docs/introduction (visited on 08/22/2022).