Contents

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].

Several approaches to educate students on secure programming are commonly considered. The first, adding it as a material to existing classes. However, if the principles and practices are introduced as extra material in existing classes, the skill is atrophied, considering that whether the students write secure programs is not being examined. The second, adding a separate class that covers secure programming in an already packed curriculum. A third approach is the introduction of a "secure programming clinic" that facilitates students on the principles and

practices of writing secure software [3]. The University of North Carolina at Charlotta introduced their own tool Educational Security in the IDE, which provides students with security warnings on assignment code for integrating secure programming education. The paper states that students awareness and knowledge on secure programming increases through the usage of ESIDE [2].

To learn secure programming practices, this thesis proposes to integrate the Security Knowledge Framework into the Software Development Life Cycle with the development of a social recipe sharing app, CheFEED. SKF is an open-source flagship project from the Open Web Application Security Project. It utilizes the OWASP Application Security Verification Standard to train developers in writing secure code by design. The ASVS provides a framework of security requirements and controls for designing, developing and testing modern web applications and web services. In addition they provide the Mobile Application Security Verification Standard that focuses security requirements and controls for mobile applications. SKF in addition has developed a security expert system that generates these security requirements for a sprint during the SDLC.

The purpose of this thesis is to examine whether SKF is a good source for developers to learn about secure software development, without a security champion in the team. The thesis is conducted as a case study to see if skill and abilities in secure programming is enhanced through the development of the CheFEED application. The features of CheFEED will form the foundation to generate the security requirements needed to develop a secure application.

1.2 Scope

The thesis focuses on fundamental security concepts and principles. They will be applied by developing the CHEFEED application. CHEFEED will have an API and a hybrid native client. The application will go through a secure Software Development Life Cycle (SDLC). An important principle in information security is to find a good balance between security and productivity. Thus, a security

maturity model and a standard for writing secure code will guide the SDLC. Adding security into the SDLC will allow to have secure code by design, instead of having security as an afterthought. Furthermore, security tests will be performed in the form of penetration tests and security code reviews.

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 Structures

This section summarizes the general description of the coverage of each chapter.

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

Chapter 2

Foundation

Chapter 2 discusses the theoretical foundations and frameworks related to security. 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, cryptography, the software development life cycle and how the Open Web Application Security Project contributes to securing software.

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].

2.1.2 Secure By Design

The term 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 and implementing software with the minimal amount of vulnerabilities that an attacker can exploit [6]. 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.

2.2 Fundamentals of Information Security

Security is built on top of well established principles like the confidentiality, integratity and availability 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 3.1.1. Nonetheless, as described in Table 2.1, the confidentiality, integrity, and availability (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].

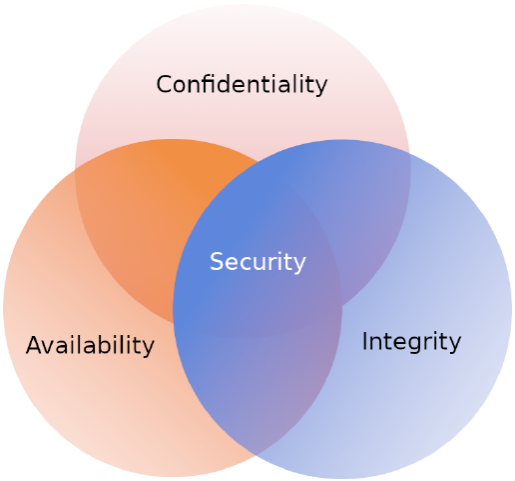

Figure 2.1: The CIA Triad

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.

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

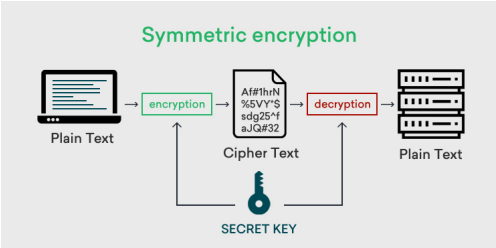

Figure 2.3: Symmetric encryption

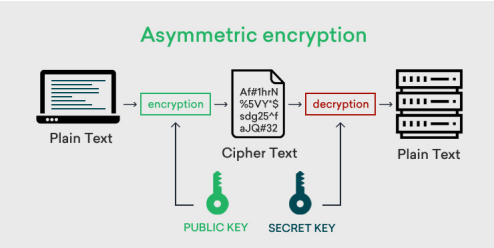

Figure 2.4: Asymmetric encryption

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.

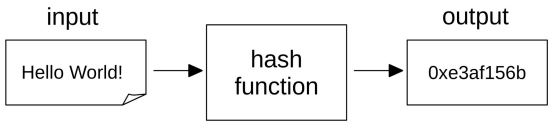

Figure 2.5: Hash function

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

Developing software is not just about the capability to write code. It is also important to know what technologies to use and what benefits they provide. This section the technology stack that will be utilized to developed CheFeed. Furthermore, this section discusses the software development life cycle.

2.3.1 Software Application 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*" [11]. 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. When implemented correctly, any organization, team, developer or operation

team will benefit from the use Docker [12].

NoSQL Databases

FastAPI

Hybrid Applications

2.3.3 The Software Development Life Cycle

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 [13]:

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

The software development life cycle (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 [14].

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 2.6. 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 untill late during the life cycle and it is also an inadequate model for large projects [14]. 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 [15].

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. Each SDLC model comes with their own advantages and disadvantages, although this thesis will not be discussing those in depth. 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.

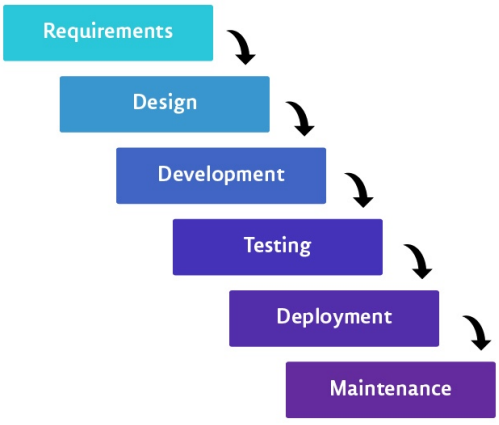

Figure 2.6: Software Development Life Cycle - Waterfall Model

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.

2.4 OWASP - Security with Open Source Tools

Without the proper tools it is much more difficult to defend against cyberattacks, and as discussed in Section 3.1.1 security has several challenges. The Open Web Application Security Project (OWASP) is a non-profit foundation dedicated on improving the security of software through its open source software projects led by the community from all over the world. In addition, OWASP hosts local and global conferences to improve software security [16]. For two decades OWASP has provided the tools needed to better secure the software with projects like the ASVS, SKF, SAMM and many more.

Chapter 3

Problem Analysis

Chapter 3 starts with describing CheFeed and analysis the problems related to security and the development of CheFeed. An overview of the features and it's potential security vulnerability are described in detail to better understand the security risk. Additionally, this chapter discusses non-functional requirements like the usage of technologies like Docker and NoSQL databases and their benefits and security risks that come along with them.

3.1 The State of Software Security

3.1.1 Challenges in Security

Computers have evolved drastically from the mechanical calculating machines they once were when first introduced in the 19th century. Today, computers are powerful machines that allows us to surf the internet, run games, and stream multimedia. Everything involves system and software technology that constantly is processing information. Important and sensitive information are controlled, managed or either form part of the cyberspace. Likewise, important and sensitive information can be exploited as well in this space [17]. Moreover, it has become very simple to stage an attack. A sophisticated attack can be constructed that affect millions of computers worldwide by an attacker with tutorials found online

together with their own knowledge and resources.

In addition, many organizations perceive security as a hindrance to productivity. It is not uncommon for discussions concerning security to be avoided among business leaders and IT personnel. Errors made by developers might be costly and might endanger everyone who trust the software they build. Though the stakes in security are high, developers perceive security as a secondary concern [18].

Even so, security can be complicated without the appropriate approach. As the level of security is increased, the level of productivity is usually decreased. Therefore, the responsibility of a security plan is to balance between insurance, usability, and cost. Moreover, in what manner the level of security relates to the value of the item being secured needs to be taken into consideration, when an asset, system, or environment is secured.

Knowledge on how to stay secure changes at a much slower pace in contrast to the increasingly accelerated rate technology changes. As a consequence, security does not always keep up with the changes. At the same time, acquiring a decent understanding of the fundamentals of information security will provide the foundation needed to manage changes as they come along.

3.2 Software Security Engineering Solutions

The need for security-enabled software has introduced several methodologies, standards and tools that can be integrated to the SDLC. This section discusses the different solutions to develop secure software. A great source for developing secure software is OWASP. Although OWASP has many projects, this section will analyze and compare only several of the flagship projects.

3.2.1 The Top Ten

The OWASP Top Ten is one of the most popular OWASP projects and is recognized by many developers as one of the first steps in producing more secure code. Table 3.1 describes the latest installment of the OWASP Top Ten list. The risks are categorized by the amount of applications tested for a given year (starting in 2017), and the amount of applications with at least one instance of a common weakness enumeration (CWE). This approach provides insight to the OWASP Top Ten team how common each CWE is within the selected applications [19].

Although the document exist to bring awareness to the most common vulnerabilities, it has not stopped organizations to use it as a security standard. However, the Top 10 is just the bare minimum of the software security requirements an application should implement. In addition, the risk characteristics of the Top 10 also makes it difficult to thoroughly detect, test or protect against [20].

Table 3.1: Top 10 Web Application Security Risks 2021

A01:2021	Broken Access Control
A02:2021	Cryptographic Failures
A03:2021	Injection
A04:2021	Insecure Design
A05:2021	Security Misconfiguration
A06:2021	Vulnerable and Outdated Components
A07:2021	Identification and Authentication Failures
A08:2021	Software and Data Integrity Failures
A09:2021	Security Logging and Monitoring Failures
A10:2021	Server-Side Request Forgery (SSRF)

3.2.2 The Application Security Verification Standard

The Application Security Verification Standard (ASVS) is another flagship project from OWASP that provides an actual standard for application security. The ASVS can be utilized by architects, developers, testers, security professionals, tool vendors, and consumers to define, develop, test and verify secure applications. The most recent version is 4.0, which had a significant change with the

adoption of NIST 8000-63-3 Digital Identity Guidelines. The two main objectives of the ASVS are:

- to help organizations develop and maintain secure applications
- to allow security service vendors, security tools vendors, and consumers to align their requirements and offerings

Applications have different security needs and different applications do not need the same level of security. Therefore, the ASVS introduces three security verification levels, each level increasing in complexity. Each level in comprises a list of security requirements that can also be mapped to security-specific features and capabilities that should be built into software as is displayed in Figure 3.1.

- **Level 1**. Low assurance level, and is completely penetration testable.
- **Level 2**. For applications that contain sensitive data, which requires protection and is the recommended level for most apps
- **Level 3**. For the most critical applications - applications that perform high value transactions, contain sensitive medical data, or any application that requires the highest level of trust.

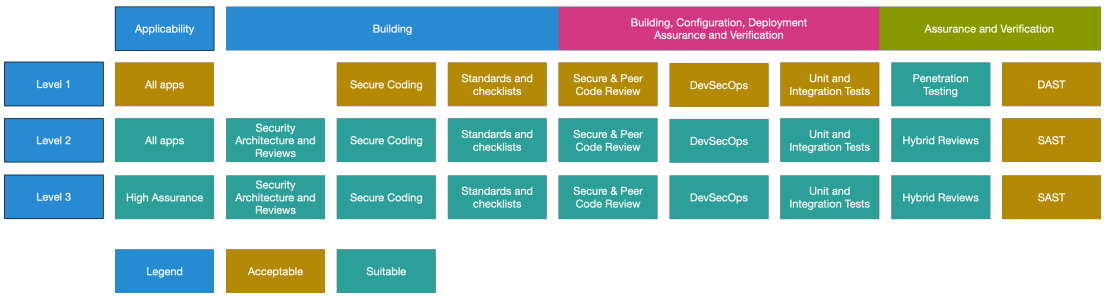

Figure 3.1: OWASP Application Security Verification Standard 4.0 Levels

Unlike Level 2 and Level 3, the security requirements in Level 1 are completely penetration testable. However, in reality it is strongly encouraged to use a broad range of security assurance and verification. Black box testing has proven to be ineffective in identifying critical security issues that led directly to ever more

massive breaches [16]. The usage of security tools like DAST and SAST during the SDLC allows detecting security issues to be found that should not be there in the first place.

ASVS Level 1

The bare minimum any application should achieve is that of Level 1, and is useful when the application is not dealing with sensitive data. It sufficiently defends against application security vulnerabilities that are simple to discover by attackers using simple and low level techniques, and are included in the OWASP Top 10 and comparable checklists as well.

ASVS Level 2

Most applications today should achieve Level 2, also known as the standard level. Applications achieving Level 2 are generally those that deal with crucial business-to-business transactions or industries where sensitive assets and the integrity is critical to the business.

ASVS Level 3

Security Knowledge Framework

Software Assurance Maturity Model

Whilst the Top 10 functions as an awareness document for security vulnerabilities, and projects like the ASVS and SKF help to identify the security requirements for applications, they do not provide the means to effectively measure the security requirements. The Software Assurance Maturity Model (SAMM) is a tool that can be utilized to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organizations, and can be integrated into the SDLC. An overview of the model can be seen in Table 3.2. It describes the four crucial Business Functions together with the three Security Practices that are related to them. Small, medium, large organizations

or even individual projects can benefit from the model since SAMM is defined with flexibility in mind. The principles that SAMM was built on are:

- An organization's behavior changes slowly over time
- No single recipe exist that is applicable to all organizations
- Guidance related to security activities must be prescriptive

Table 3.2: SAMM Model Overview

Governance	**Implementation**	**Verification**	**Operations**
Strategy and Metrics	Secure Build	Architecture Assessment	Incident Management
Policy and Compliance	Secure Deployment	Requirements-driven Testing	Environment Management
Education and Guidance	Defect Management	Security Testing	Operational Management

3.3 Proposed Solution

The ASVS provides a great set of security standards that can be mapped to each functional requirement of the software that is being developed. Together with the security expert system from SKF each sprint of the SDLC can have security first approach.

Chapter 4

Software Requirements Specification

Chapter 4 discusses the functional description and technical description for Chefeed. The solution design will be explained in the form of diagrams, tables or figures. Moreover, the Chefeed API will be described in depth as well as the security requirements.

4.1 Requirements Specification

This section describes the product features using high-level statements of what the software must do and how it should function focusing on the user needs.

4.1.1 Functional Requirements

The functional requirements are described utilizing user stories and the requirements they must fulfill in order to be accepted. User stories put the end-user first, thus effectively specifying what a crucial functional requirement is for Chefeed. Table 4.1 states the user stories and their acceptance criteria. Each user story is actionable sprint within the SDLC.

Table 4.1: User stories and requirements

	User story	**Acceptance Criteria**
Registration	As a user, I want to be able to register an account	The application must allow users to register using an email address and password
Authentication	As a user, I want to be able to authenticate with the registered email and password	The application must allow users to authenticate with the confirmed email address and set password
Password reset	As a user, I want to be able to reset my password	The application must allow users to reset their password by clicking *"forget password"*
Delete account	As a user, I want to be able to delete my account	The application must allow users to delete their registered account
Create recipe	As a user, I want to be able to post recipes	The application must allow authenticated users to post recipes
Retrieve recipes	As a user, I want to be able to read recipes	The application must allow both authenticated and not authenticated user to read recipes
Update recipe	As a user, I want to be able to update my created recipes	The application must allow authenticated and authorized users to update their existing recipes
Delete recipe	As a user, I want to be able to delete my created recipes	The application must allow authenticated and authorized users to delete their own recipes
Comment on recipes	As a user, I want to be ablte to comment on recipes	The application must allow authenticated users to comment on recipes
Update comment	As a user, I want to be able to update my posted comment	The application must allow authenticated and authorized users to update their comments
Delete comment	As a user, I want to be able to delete my posted comments	The application must allow authenticated and authorized users to delete their created comments

4.2 API Design

The CheFeed API is organized around REST. This section describes how the API is designed around the resources.

4.2.1 Core Resources

- Recipes
- Categories
- Ingredients
- Bookmarks
- Reviews (comments)

Recipes

This object represents a Recipe object that belongs to a User object. Reading recipes does not require authentication, thus users without accounts are authorized to view recipes. Table 4.2 describes all the resources related to the Recipe object and what HTTP request methods are available.

Listing 4.1: The Recipe example object

```
{
    "id": "",
    "title": "Rendang",
    "description": "The world's tastiest beef stew",
    "cooking_time": 3,
    "image": "path/to/image",
    "video": "path/to/video",
    "created_at": "2022-08-02",
    "updated_at": ""
}
```

Categories

This object represents a Category object and the resources are described in Table 4.3.

Table 4.2: Recipe API endpoints

Method	**URL**
GET	*/api/v1/recipes/*
GET	*/api/v1/recipe/**:id***
CREATE	*/api/v1/recipe/create*
PUT	*/api/v1/recipe/update/**:id***
DELETE	*/api/v1/recipe/delete/**:id***

Table 4.3: Category API endpoints

Method	**URL**
GET	*/api/v1/categories/*
GET	*/api/v1/category/**:id***
CREATE	*/api/v1/category/create*
PUT	*/api/v1/category/update/**:id***
DELETE	*/api/v1/category/delete/**:id***

Ingredients

This object represents an Ingredient object and the endpoints for this resouce are described in Table 4.4.

Table 4.4: Ingredient API endpoints

Method	**URL**
GET	*/api/v1/ingredients/*
GET	*/api/v1/ingredient/**:id***
CREATE	*/api/v1/ingredient/create*
PUT	*/api/v1/ingredient/update/**:id***
DELETE	*/api/v1/ingredient/delete/**:id***

Bookmarks

This objects represents a Bookmark object and the endpoints for this resource are described in Table 4.5.

Table 4.5: Bookmark API endpoints

Method	**URL**
GET	*/api/v1/bookmarks/*
GET	*/api/v1/bookmark/**:id***
CREATE	*/api/v1/bookmark/create*
PUT	*/api/v1/bookmark/update/**:id***
DELETE	*/api/v1/bookmark/delete/**:id***

Reviews

This object represents a Review object and the endpoints for this resource are described in Table 4.6. The Review object can also be referred to as the Comment object as they both describe the same concept within the application.

Table 4.6: Review API endpoints

Method	**URL**
GET	*/api/v1/reviews/*
GET	*/api/v1/review/**:id***
CREATE	*/api/v1/review/create*
PUT	*/api/v1/review/update/**:id***
DELETE	*/api/v1/review/delete/**:id***

4.2.2 Authentication

Chefeed handles its authentication mechanism and user management through the FastAPI Users library, a ready-to-use library for FastAPI.

4.2.3 Errors

4.2.4 Versioning

Changes are inevitable in software development, and it must be managed well. Without proper management, changes can threaten the client integrity. APIs only are required to up-versioned when a breaking change is introduced. For instance changes in the format of the response data for one or more requests, a change in the request or response type, or whenever any part of the API is removed. Chefeeds API follows the versioning method similar to that of Stripe.

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] Matt Bishop et al. "Evaluating Secure Programming Knowledge". In: *Information Security Education for a Global Digital Society*. Springer International Publishing, 2017, pp. 51–62. DOI: 10.1007/978-3-319-58553-6_5. URL: https://doi.org/10.1007%2F978-3-319-58553-6_5.

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

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

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

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

[14] 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.

[15] 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.

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

[17] Yuchong Li and Qinghui Liu. "A comprehensive review study of cyber-attacks and cyber security Emerging trends and recent developments". In: *Energy Reports* 7 (Nov. 2021), pp. 8176–8186. DOI: 10.1016/j.egyr.2021.08.126. URL: https://doi.org/10.1016%2Fj.egyr.2021.08.126.

[18] Tamara Lopez et al. ""Hopefully We Are Mostly Secure": Views on Secure Code in Professional Practice". In: *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, May 2019. DOI: 10.1109/chase.2019.00023. URL: https://doi.org/10.1109%2Fchase.2019.00023.

[19] *OWASP Top 10 2021*. URL: https://owasp.org/Top10/A00_2021_Introduction.

[20] *How to use the OWASP Top 10 as a standard*. URL: https://owasp.org/Top10/A00_2021_How_to_use_the_OWASP_Top_10_as_a_standard.