Contents

1 Introduction **3**

1.1 Background . . . 3

1.2 Scope . . . 4

1.3 Aims and Benefits . . . 4

1.3.1 Aims . . . 4

1.3.2 Benefits . . . 5

1.4 Structures . . . 5

2 Foundation **6**

2.1 Information Security . . . 6

2.1.1 Defining Information Security . . . 6

2.1.2 Secure By Design . . . 7

2.2 Fundamentals of Information Security . . . 7

2.2.1 The Confidentiality, Integrity, and Availability Triad . . . 8

2.2.2 The Gold Standard . . . 9

2.2.3 Cryptography . . . 11

2.3 Software Development . . . 14

2.3.1 Software Application Architecture . . . 14

2.3.2 Technology Stack . . . 14

2.3.3 The Software Development Life Cycle . . . 15

2.4 OWASP - Security with Open Source Tools . . . 17

3 Problem Analysis **18**

3.1 Proposed Solution 18

3.1.1 Security Requirement Analysis 18

4 Software Security Design **21**

4.1 Security Requirement Analysis 21

4.1.1 Architecture, Design and Threat Modeling Requirements . 21

4.1.2 Data Storage and Privacy Requirements 22

4.1.3 Authentication and Session Management 22

Chapter 1

Introduction

1.1 Background

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

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

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

1.2 Scope

This study covers the implementation of SKF for developing secure mobile applications. SKF provides three security maturity levels from ASVS and MASVS. This study limits itself to achieve level 1 for "Authentication an Session Management Requirements", "Data Storage and Privacy Requirementsqqq:"

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 **??**. 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].

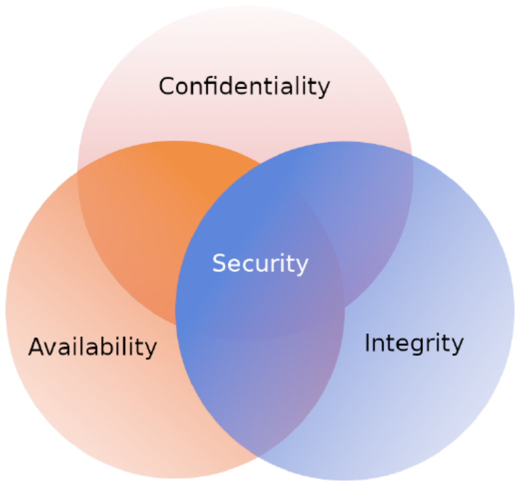


Figure 2.1: The CIA Triad

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.

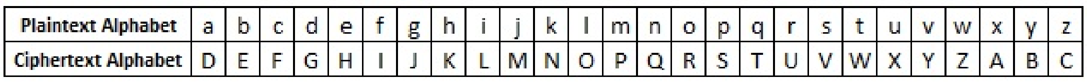
Auditing

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

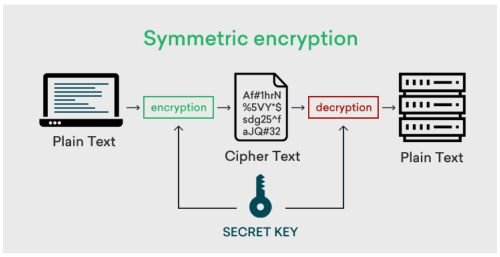


Figure 2.3: Symmetric encryption

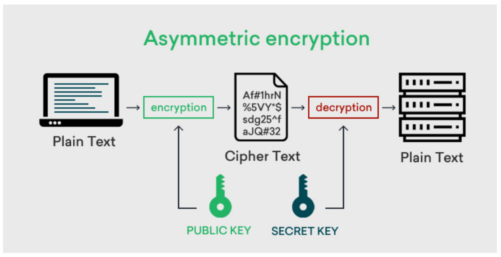


Figure 2.4: Asymmetric encryption

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.

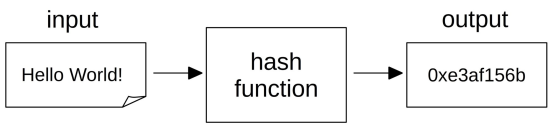


Figure 2.5: Hash function

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

The Waterfall Model 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.

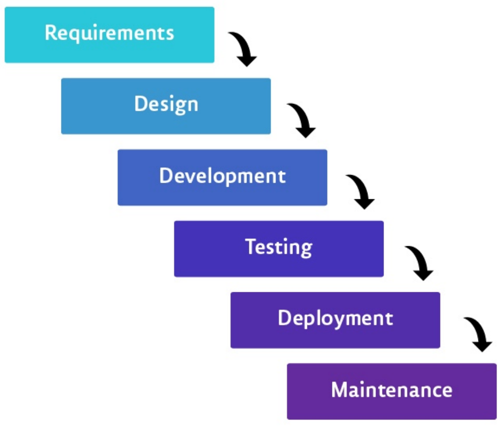


Figure 2.6: Software Development Life Cycle - Waterfall Model

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 **??** 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 [3]. 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

3.1 Proposed Solution

The solution this thesis proposes is implementing SKFs security expert system to develop secure mobile applications. Each sprint is configured with the level 1 type security verification requirements generated from the MASVS. The following security controls have been included:

- **V1**: Architecture, Design and Threat Modeling Requirements
- **V2**: Data Storage and Privacy Requirements
- **V4**: Authentication and Session Management Requirements

3.1.1 Security Requirement Analysis

Architecture, Design and Threat Modeling Requirements

CheFeed follows a three-tier software architecture as illustrated in Figure 3.1. Therefore, appropriate security standards must be implemented to all services. The category "Architecture, Design, and Threat Modeling Requirements" ensures that security is addressed when planning the architecture of the mobile app. Table 3.1 describes the sprint configuration for this category. The sprint results described in Table 3.2 verification requirements mapped to their MSTG-ID.

Figure 3.1: System Architecture

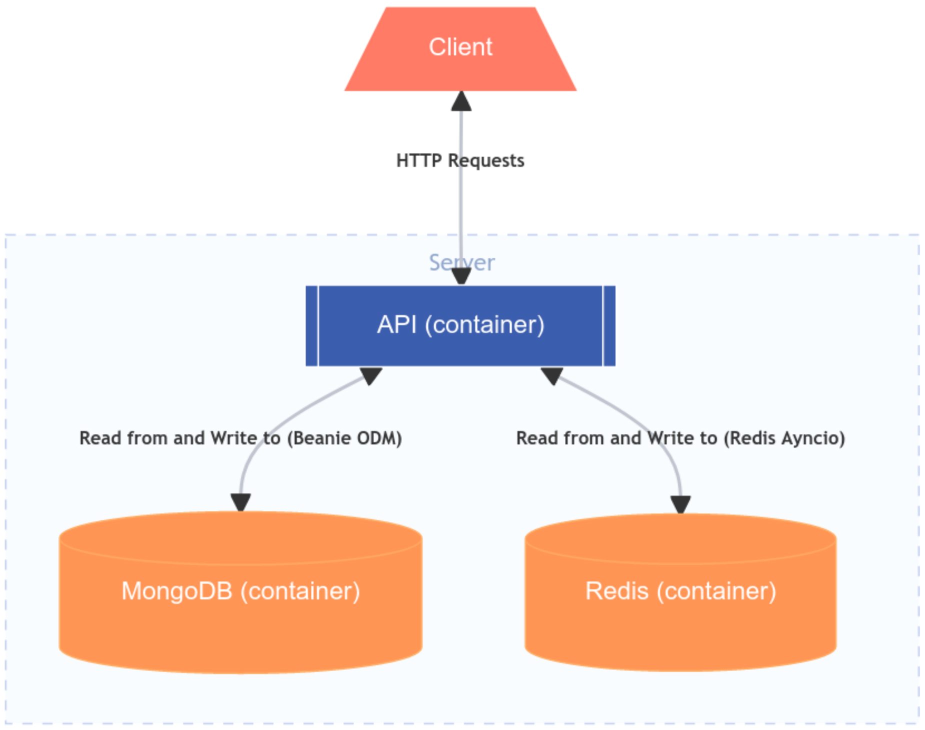


Table 3.1: V1 Sprint Configuration

Description	**Configuration**
Centralized security requirements	Yes
Does your application accept and process any inputs?	Yes
Does your application need to comply with some laws or regulations?	No
Does your application need to encrypt data?	Yes
Does you application need to have a responsible disclosure policy?	No
Update policy requirements	No
Does your application need to provide data protection	Yes
Secure software development life cycle requirements	Yes

Table 3.2: Architecture, Design and Threat Modeling Requirements sprint results

MSTG-ID	**Description**
1.1	All app components are identified and known to be needed
1.2	Security controls are never enforced only on the client side, but on the respective remote endpoints
1.3	A high-level architecture for mobile app and all connected remote services has been defined and security has been addressed in that architecture
1.4	Data considered sensitive in the context of the mobile app is clearly identified

Data Storage and Privacy Requirements

Personally identifiable information (PII) such as email address or the users full name can be used for identity theft. Therefore, CheFeed must ensure sensitive data is protected.

Authentication and Session Management Requirements

Chapter 4

Software Security Design

Bibliography

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

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

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

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

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

[6] 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`.