Security Code Review
Added Value or Wasted Money?

Brinio Hond MSc
briniohond@gmail.com
Supervisor: Dr. René Matthijsse RE

September 26, 2014
Abstract

Not a day goes by without at least one news article about a digital attack on the Information Systems (IS) service at some organisation. Digital attacks have become more and more prevalent over the years. In order to defend against digital attacks on their software organisations can employ different methods to verify the security of this software. One of these methods is the security code review, in which the source code of the software is reviewed by an expert to identify security vulnerabilities.

In this thesis we look into the added value of these security code reviews for the security of software. Security code reviews are not (yet) used often in practice and the value of the method has not been clearly defined. Through theory study and verification of our findings with experts we show that security code reviews can certainly add value for the security of almost all software. For small and business critical applications a security code review can give an organisation sufficient assurance that the software is secure. For larger applications a combined approach with penetration testing provides good results for reasonable costs. Therefore we believe that whenever the security of software is questioned, security code review can give valuable insights in the state of the software’s security.
## Contents

1 Introduction .......................................................... 4  
1.1 Background .......................................................... 4  
1.2 Research question ..................................................... 5  
1.3 Research methodology ................................................. 6  
1.4 Relevance for the IS auditor .......................................... 6  
1.5 Previous work ......................................................... 7  
1.6 Outline ................................................................. 8  

2 Software Security ....................................................... 9  
2.1 Information Security .................................................. 9  
2.1.1 CIA Triad .......................................................... 10  
2.1.2 Data Ownership .................................................... 10  
2.1.3 Other Information Security Properties .......................... 11  
2.2 Software Security ..................................................... 11  
2.2.1 Functionality vs. Security ......................................... 12  
2.2.2 Defining Software Security ....................................... 12  
2.3 Secure Software Development ....................................... 14  
2.3.1 Systems Development Life Cycle ................................. 14  
2.3.2 Securing the Systems Development Life Cycle ................. 16  

3 Verifying Software Security ............................................. 19  
3.1 Software Security Verification Methods ............................ 19  
3.1.1 Software Development Process Review ......................... 19  
3.1.2 Software Certification ............................................. 20  
3.1.3 Static Analysis .................................................... 21  
3.1.4 Dynamic Analysis .................................................. 21  
3.1.5 Fuzz testing ....................................................... 21  
3.1.6 Penetration Testing ............................................... 22  
3.1.7 Security Code Review ............................................. 22  
3.2 Security Code Review ................................................. 22  
3.2.1 OWASP Security Code Review Process ......................... 23  
3.2.2 Security Innovations Security Code Review Process ........... 24  
3.2.3 Infosec Institute Security Code Review Process ............... 25  
3.2.4 Generic Security Code Review Process .......................... 25  
3.3 Advantages and Disadvantages of Security Code Reviews ........ 30  
3.3.1 Security Code Review Advantages ............................... 30  
3.3.2 Security Code Review Disadvantages ............................ 31
Chapter 1

Introduction

1.1 Background

Not a day goes by without at least one news article about a digital attack on the Information Systems (IS) service at some organisation [36]. Digital attacks can lead to a digital security breach or IS outage, which disrupts the business. Digital security breaches and IS outages have become so prevalent, that it almost seems as if an organisation that has not been hacked is not worthy of existing.

The impact of a digital security breach or IS outage can vary wildly, but usually it causes one or more of the following: loss of revenue, loss of time to fix the issues, reputational damages, potential law suits and/or loss of business productivity.

There are many causes for digital security breaches or disruption in the IS service. Nevertheless, these causes can be aggregated in three different domains. These domains together comprise the digital security of an organisation and are described below:

- **People**, e.g. lack of user awareness, such as the use of weak passwords.
- **Process**, e.g. failing IT General Controls (ITGC), such as inadequate change or patch management processes.
- **Technology**, e.g. technical vulnerabilities in software, such as SQL injection on a website.

Organisations can take different measures to prevent digital attacks, aligned to the three domains: people, process and technology. In this thesis the focus is on the technology domain, and more specifically the software that implements the technological security measures to try to prevent digital security attacks.

Software is used by every organisation, both to complete general tasks (emailing, word processing and web browsing) and to complete organisation specific tasks (for example performing a specific payment process). Some of this software will have been developed by the organisation using it, while other software is developed by a third party. Because many business (critical) processes depend on the correct and secure functioning of this software, organisations take measures to limit risks of failing software or a digital security breach.
Each measure that can be taken by an organisation generally fulfils at least one of three different purposes: 1) preventing software failures or digital security breaches from happening, 2) detecting a potential digital security breach and 3) responding to a potential digital security breach. These three purposes are in no way limited to software security or IS. An analogy to security in the physical world for these three types of measures would be locks (preventive), surveillance cameras with a monitoring team actively monitoring what happens (detective) and a security team that will act when suspicious activity has been detected by the monitoring team (responsive).

In order to evaluate the effectiveness of the implemented security measures an organisation can use an Information Systems audit. Although there is no single universal definition of IS audits (or Electronic Data Processing (EDP) audits as they were formerly called), Weber gave an accurate definition in 1988:

**Information Systems (IS) audit:** The process of collecting and evaluating evidence to determine whether a computer system (information system) safeguards assets, maintains data integrity, achieves organisational goals effectively and consumes resources effectively [42].

Software is an example of an information system and hence, it can be the subject of an IS audit. In this thesis we look at the effectiveness of one method to perform an IS audit on software: the security code review [25]. This method is a more focused and specific type of the generic code review. Security code review is defined by the Open Web Application Security Project (OWASP) as follows:

**Security code review:** The process of auditing the source code for an application to verify that the proper security controls are present, that they work as intended, and that they have been invoked in all the right places [35].

### 1.2 Research question

Even though the concept of security code reviews is not new (from an IT perspective), it does not seem to be used very often in practice. This is noteworthy since the reliance of organisations on software, and therefore the correct working of the software, is only increasing. This observation has formed the basis for this thesis and led to the following research question:

**What are the considerations to perform a security code review to verify software security?**

In order to answer this research question the following sub-questions are defined:

1. What is software security and which verification methods exist?
2. What is security code review and when can it theoretically be applied?
3. What are the main elements and characteristics of security code reviews in a practical situation and what are the main values?
4. In which situations do security code reviews add value to software security?
1.3 Research methodology

In order to answer the research question a three phase approach is used. The first phase is to perform a theoretical study to discover what precisely security code reviews are and when they can be applied. In the second phase subject matter experts are interviewed to discover more about security code reviews in practical situations and what the main values and costs are. In the last phase the results of the theoretical and empirical study will be combined in order to find out what the considerations are for security code review to verify software security.

1.4 Relevance for the IS auditor

It is not a secret that many organisations struggle with IS security [36]. As already discussed in the background section organisations can request an IS auditor to help them gain assurance on IS security.

In general any type of auditor can give two different levels of assurance, limited assurance or reasonable assurance. The theoretical third and highest level (absolute assurance) can generally not be given, e.g. due to inherent limitations of internal controls and the use of sampling [30]. The difference between limited and reasonable assurance lies in the confidence the auditor has attained during the audit that the tested controls are effective. This is directly related to the depth of the audit. The level of assurance can always be deduced from the conclusions; limited assurance is formulated negatively (for example “we did not identify anything that shows the controls are not effective”\(^1\)), while reasonable assurance is formulated positively (e.g. “we are of the opinion that the controls are effective”\(^1\)).

The two different assurance levels are no different for the IS auditor. Each assurance level requires a different approach, especially when the object of the audit is software. Different software security testing methods are available, some of which will be discussed later in this thesis, and different methods can reach different levels of assurance. For now it suffices to look at an example to be able to give reasonable assurance on the security of software. One of the most common and prevalent vulnerabilities for many years has been Injection [33]. A very common and effective method to prevent SQL Injection is to use parameterised database queries instead of dynamically creating the queries [32].

How can an IS auditor give reasonable assurance on the control objective to always use parameterised database queries and prevent the dynamic creation of queries?

In order to be able to give reasonable assurance the IS auditor has to be able to validate that parameterised queries are used. Testing the application, either with automated tools or manually, for SQL Injection vulnerabilities is not enough for this. If the tests would not show any SQL Injection the IS auditor still only has limited assurance, because the auditor has only shown there is no reason to believe that parameterised queries are not used. But it could be that the test set was not extensive enough to find an injection vulnerability, while such a vulnerability does exist because parameterised database queries

---

\(^1\) Please note that the example conclusions are simplified, as actual conclusions would also refer to the audit work that has been performed, materiality, etc.
are not used. In order to get a higher level of assurance the IS auditor has to review the source code of the software (manually or automated) and verify that parameterised queries are used, i.e. perform substantive testing. Or in other words, the IS auditor has to perform a security code review on the software to be able to give reasonable assurance on this control objective and therefore the security of the software. Without security code review the IS auditor is not able to give reasonable assurance about many control objectives that can be defined to verify software security.

1.5 Previous work

Different researchers and experts from the field of software development, code reviewing and IT consulting have already studied different aspects of security code reviews. Especially the security code review process itself has been described numerous times, for example by OWASP [34], Microsoft [24], the company Security Innovation [14] and the organisation Infosec Institute [31].

The different descriptions of the security code review process have a lot in common. They describe the different phases in the review process and give tips and tricks on what tools to use and what methods to employ. Some of them also give the advantages of using security code reviews already during the development process. For example, in [31] four benefits are mentioned: 1) effort benefit of finding bugs early in development, 2) cost benefit of finding bugs early in development, 3) compliance to certain security standards such as PCI-DSS² and 4) reputation benefit of finding bugs before production where the customers may be affected by it. However, none of these references seem to discuss what the preconditions are for security code review to really add value. They do give a lot of benefits, but seem to completely skip any drawbacks.

Another valuable resource on the topic of software security verification is the book by Dowd et al. [11]. In this book various software security verification methods are described and explained, along with clear examples to show how they could find security flaws in actual programs. Security code review is one of the methods that is extensively described. However, the book does not go into the actual use of the described security verification methods in practice. As a result it does not help to identify why security code reviews are not used so often in practice, nor to help find out when they add value to software security.

In 2009 the US Naval Sea Systems Command published a white paper that reviews different software security verification methods and tools [9]. This paper gives an overview of different tools and when they are best used. Also the benefits and drawbacks are described and presented in a table for easy comparison between different tools. It contains a whole section on source code analysis methods and tools. However, the focus of this paper is on automated tools and automated software security verification methods only. Hence it does not cover the manual aspects of security code reviews, nor does it cover all different security verification methods. Therefore the comparison is only useful if the reader is looking into purely automated analysis.

²The standard to which all organisations that accept, transmit or store credit cardholder data must comply. For more info see https://www.pcisecuritystandards.org/security_standards/
In 2013 the results of an empirical study on the effectiveness of security code reviews was published by Edmundson et al. [12]. This study looked at the results of a manual security code review by thirty independent reviewers on a small piece of code with several known vulnerabilities. Their results have shown that approximately 80% of the vulnerabilities were identified by combining the results of ten different reviewers, and this percentage increased to 95% for fifteen reviewers. These numbers do not seem very promising, but the set of reviewers included web developers that may have had little experience in security code reviews. In addition, the reviewers were only supposed to perform a manual research and do not use any automated tools to assist the discovery of vulnerabilities.

1.6 Outline

In chapter 2 information security and software security are described, including different methods to verify the security of software. In chapter 3 security code review is described in much detail, including pros and cons of security code review and preliminary conclusions. In chapter 4 the opinions of the subject matter experts are described, followed by a practical case study in chapter 5. Chapter 6 will describe the conclusions, which are based on the theory study and the expert interviews, as well as some pointers for future work.
Chapter 2

Software Security

In this chapter we define what we mean with software security. However, we will first dive deeper into the world of information security. Software processes data (or in other words information) and software security serves as purpose to protect the information that the software processes. Understanding information security is therefore a prerequisite to understanding software security.

In the remainder of the chapter we take a closer look at the software development process. This process is often referred to as the software development life cycle.

2.1 Information Security

In the current information age all organisations process a lot of information, such as customer data, employee data, payment data, intellectual property etc. This information should be secured in order to protect the business of the organisation. Information security is therefore important for all organisations, whether it is a small one-man shop or a multi-billion multinational firm.

But what is information security exactly? Because information security is so important several different standards and best practices to ensure information security exist. Probably the most well-known standard is the ISO27000 series, which describes control objectives and best practices for creating an Information Security Management System (ISMS) within an organisation. The definition for information security given within the ISO27000 standard is the following:

Information Security (ISO27000): The preservation of confidentiality, integrity and availability of information. In addition, other properties, such as authenticity, accountability, non-repudiation and reliability can also be involved [19].

This definition is very similar to the definition of information security by other bodies. For example ISACA\(^1\) defines information security as:

Information Security (ISACA): Something that ensures that within the enterprise, information is protected against disclosure

\(^1\) “An independent, nonprofit, global association, that engages in the development, adoption and use of globally accepted, industry-leading knowledge and practices for information systems” [15].
to unauthorised users (confidentiality), improper modification (integrity) and non-access when required (availability) [16].

In the next subsection we look in more detail into the three properties that are part of both definitions of information security: confidentiality, integrity and availability. We then go into more detail regarding data ownership and finally we take a quick glance at the other properties named in ISO27000: authenticity, accountability, non-repudiation and reliability.

2.1.1 CIA Triad

Both of the information security definitions talk about three security properties: 1) confidentiality, 2) integrity and 3) availability. These three properties are the heart of information security and are sometimes referred to as the CIA triad. In order to understand the definition of information security we have to further define these properties. We again refer to ISO27000 for the definitions.

Confidentiality: Property that information is not made available or disclosed to unauthorised individuals, entities, or processes [19].

Integrity: Property of protecting the accuracy and completeness of assets [19].

Availability: Property of being accessible and usable upon demand by an authorised entity [19].

These definitions are quite formal and might require reading multiple times to really understand. Therefore we also describe these properties in a simpler language, which is accurate enough to the original definitions:

- **Confidentiality.** Only authorised people have access to the data.
- **Integrity.** Only authorised people can modify the data.
- **Availability.** The data is available whenever it is needed.

For all of these properties it is important that the authorised users and availability criteria are adequately defined. This has to be done by the owner responsible for the data, as described in the next subsection.

2.1.2 Data Ownership

All information within an organisation has a purpose, whether it is a calendar event for a weekly meeting between floor workers and their manager or the detailed recipes for the drinks sold by a soda company. For each of these different bits of information there should be someone in the organisation responsible, the data owner. The data owner controls the information and not only has the ability to access, create, modify, package, derive, benefit from, sell or remove the data, but also the right to assign these privileges to others [21].

---

2It is important to note that availability does not mean the data has to be available 24/7, unless that is specifically required for this data.
The data owner has to guarantee the security of this information, so that it can be used for its purpose whenever it is needed. Or in other words, the data owner has to ensure the confidentiality, integrity and availability of the data. To do this, the data owner defines who can have what type of access to the data and when. Ideally this is done using the principle of least privilege. This means that the users of the data of the most restrictive set of privileges assigned that still let them perform their tasks. The principle of least privilege limits that probability that the confidentiality, integrity or availability are compromised, due to both limiting the number of users that have access to the data as well as limiting the actions these users can perform with the data.

2.1.3 Other Information Security Properties

In the ISO27000 standard several other information security properties are named that could also be a part of information security. These properties have in common that they are not applicable in all information security scenarios, but can be very important in certain specific scenarios. In this subsection we explain the properties and when they could be applicable.

- **Authenticity.** Property that an entity is what it claims to be. This property is important in a secure communication scenario between two systems, two persons or a person and a system. For example, when making an online purchase in a web shop with a credit card it is for the purchaser very important that the web shop is authentic, i.e. that the credit card details are sent to the web shop and not an adversary.

- **Accountability.** Assignment of actions and decisions to an entity. This property is important in order to be able to check at a later date who was accountable for a certain action or change in the information. This allows an organisation to take appropriate actions against the accountable person should an incorrect change have been performed and detected.

- **Non-repudiation.** Ability to prove the occurrence of a claimed event or action and its originating entities. This property is important whenever a contract is signed. When non-repudiation of the information is guaranteed it is not possible for either of the parties involved in the contract to claim afterwards that they did not sign the contract. Both authenticity (of the parties signing the contract) and integrity (of the contract itself) are important prerequisites for being able to guarantee non-repudiation.

- **Reliability.** Property of consistent intended behaviour and results. Information that is reliable can also be called trustworthy. For example, the reliability of the address of a customer is very important for a web shop to prevent delivering goods to the wrong persons.

2.2 Software Security

In the previous section we have defined information security. In this section we will define software security and explain the link with information security.

All software is used to process information. The data owner of this information has set certain requirements to preserve the confidentiality, integrity
and availability, and therefore the security of the information. Secure software abides to the information security requirements set on the processed data. But this is unfortunately not as easy as it sounds.

In this section we first explain why software security is not trivial by comparing security of software with functionality of software. Then we provide a formal definition of software security.

2.2.1 Functionality vs. Security

Software is built with a specific purpose, to perform a certain function. Already back in the 1950s it was considered obvious that one of the first steps in the development process is to determine the functional requirements for the software: what should it do? Even though this is far from a simple task, the result is an idea of the functionality that has to be provided by the software. Once the software is finished it should be functional, i.e. it should perform the required functions. If the software does not do what it should the users will complain and it will have to be fixed (or not, and then it will become one of the many failed IT projects).

Software security can be considered the complete opposite of software functionality. Where functional software does what it should do, secure software does not do what it should not do. As an example, let us look at a simple program with one function: it should give read access to all data to authorised users. When the software actually gives authorised users access to the data it does what it should do and is therefore functionally correct. Testing whether the software is functionally correct is fairly easy, as it is possible to verify for different authorised users if they can access all data. But is it also secure?

The easiest way to implement the above requirement is to simply give everyone full access to all data. Then authorised users have read access to all data and voilà, the program does what it should do. But this implementation is obviously not secure, as it compromises not only the confidentiality of the data by granting unauthorised users read access, but also the integrity and availability of the data by granting more than just read access.

A secure implementation on the other hand would not do what it should not do. It would deny access for all but authorised users. It would also not grant anyone write access. And it would still perform this way when an authorised and unauthorised user use the software simultaneously (granting access only to the authorised user). And it would not transfer money from the user’s bank account, nor would it launch missiles. And and and...

This example illustrates why building secure software is difficult. The functional requirements are often specified, limited and can be tested for, while security requirements are often implicit and one can think of an unlimited amount of security requirements for any software. As a result it is much harder to test for the security requirements than it is for functional requirements.

2.2.2 Defining Software Security

The example in the previous subsection has given an idea of what software security is. Unfortunately not many researchers have formally defined this term.

3For simplicity’s sake we do not regard how the users are authenticated and how the authorisations of the user are checked, we just assume that this is taken care of externally.
One that did was McGraw, who defined software security in 2004 as the following: “The idea of engineering software so that it continues to function correctly under malicious attack” [23].

This definition includes several important aspects of software security. First of all software security is mostly an “idea” and not something you can touch. It is a mindset that is required when developing software to end up with secure software. The second important aspect of this definition is that with software security the software will “continue to function correctly”. Correctly means that the software does not stop doing what it is supposed to do, i.e. it remains to function like specified. Lastly, the definition mentions “malicious attacks”, which has something to do with an adversary that attempts to break or crack the software.

However, we believe that the definition from McGraw is not complete. There are two aspects of software security that are not adequately covered in this definition. We will therefore try to refine it to a more complete definition.

1. The definition of McGraw seems to indicate that software security is only relevant when the software is “under malicious attack”. We would argue that this is not the only scenario in which software security counts. Besides a malicious attacker there is another threat to the confidentiality, integrity and availability of the data processed by software. This threat has been described quite nicely in a quote attributed to Albert Einstein:

   Only two things are infinite: the universe and human stupidity; and I'm not sure about the universe.
   
   Albert Einstein

People make mistakes and the software should help prevent those mistakes from making undesired changes to the integrity and availability of the data. For example, a program that has a button to remove all data should at least ask for confirmation from the user before actually removing all data. This to prevent a simple mistake by the user from compromising the availability of the data. Preventing mistakes by clumsy users is also part of software security, as it helps to guarantee the integrity and availability of the data that is processed by the software.

We believe that secure software should not only continue to function when under malicious attack, but actually in all circumstances.

2. There is yet another aspect of software security that is not covered by the definition of McGraw. This aspect could be called “functional security” and is something that is quite often forgotten or ignored. Functional security refers to functional requirements with a security focus, such as being able to set a password policy in the software or being able to collect logging information from the software in a central location in a standard format. Functional security requirements are often necessary to comply with organisational security requirements as set out in the Information Security Policy or related documents, as well as to future changes to these requirements. For instance, software may set a password policy that is according to the current organisational password policy, but if this password policy cannot be changed in the software (except by the developers, requiring a
new release and associated costs) it may not comply to a future organisational password policy. Functional security requirements therefore ensure compliance with current and future organisational policies.

We believe that secure software also implements functional security requirements as set forth in the design phase.

We have defined two additions to the definition of software security by McGraw in [23]. Incorporating these changes into the definition of software security given by McGraw gives us the following definition for software security. This is the definition that we will work with in this research:

**Software Security:** The idea of engineering software so that it fulfills functional security requirements set by the user and continues to function correctly under all circumstances.

### 2.3 Secure Software Development

In the previous section we claimed that already in the 1950s defining functional requirements for software was an obvious step. The software development process back then looked quite different from what it is now. In the current age most software is developed using a Systems Development Life Cycle (SDLC) that describes the entire life span of the system (which is software in our case), from cradle (defining the need for the software) to grave (abandoning the software for a new product).

In this section we first look at two different systems development life cycles and extract the bare essentials from them. Then we describe how software security fits in the systems development life cycle to improve the security of the developed software.

#### 2.3.1 Systems Development Life Cycle

Different systems development life cycles exist and are used every day. The most important differences between these life cycles have to do with the software development process that is used to actually design, build and implement the software. When the traditional waterfall method is used to develop the software, a more traditional systems development life cycle is used, such as the one shown in figure 2.1. When instead an agile approach is used to develop the software, the iterations from the agile aproach are reflected in the systems development life cycle, as shown in figure 2.2.

Abstracting from the software development process in the above two example life cycles leaves us with a basic, four phase systems development life cycle that is applicable for all software development processes. This cycle is shown in figure 2.3 and the different phases are described below:

1. **Initiation phase.** In this phase the need for the software and its high-level purpose are defined and documented. At the end of this phase it is clear that building the system is economically and practically feasible.
Figure 2.1: The systems development life cycle for software developed using the traditional waterfall method [40].

Figure 2.2: The systems development life cycle for software developed using an agile approach, taken from: [http://www.ambysoft.com/essays/agileLifecycle.html](http://www.ambysoft.com/essays/agileLifecycle.html)
2. Implementation & Deployment phase. In this phase the software is built, using the desired software development process. This phase includes several subphases, which can be performed only once (in the waterfall model) or can be iterated over (in an agile approach):

(a) requirements analysis;
(b) design;
(c) implementation;
(d) testing;
(e) deployment.

At the end of this phase the software is finished and deployed in the production environment.

3. Operations & Maintenance phase. In this phase the software is operated and fulfills its purpose. Any necessary or desired changes to the system are developed and tested in separate environments before they are implemented in the production environment.

4. Disposal phase. Once the system has outlived its purpose it is disposed of. A system has outlived its purpose when the process it implemented has been changed or when a new system has been implemented to perform this process.

2.3.2 Securing the Systems Development Life Cycle

Now that we understand the systems development life cycle we can look into building security into this life cycle process. Different methods exist to make the systems development life cycle into a secure development life cycle. In this research we look at a relatively basic methodology by OWASP [27].

An overview of the OWASP methodology is shown in figure 2.4. The filled arrows are the different phases in the systems development life cycle, while the gradient arrows describe the security activities that should be performed in each phase to secure the systems development life cycle. Below all eight security activities are shortly described:

1. Application Classification. Performed in the Initiation phase. In this activity the application characteristics that will impact security are defined in order to classify the application’s security level. The classification determines which additional security activities the organisations deems necessary to be confident that this software is secure enough. Application characteristics can include for example whether the application is internal or external (i.e. Internet) facing and the sensitivity level of the data processed by the software.
2. **Application Security Workshops.** Performed at any point in the life cycle. The goal of the workshops is to train all personnel involved in the life cycle (including designers, developers, testers and managers) to understand software security and why it is important. It also helps them understand what they can and should do to improve software security.

3. **Application Security Requirements.** Performed in the Requirements subphase. Besides defining functional requirements for the software, also security requirements should be defined. This helps understanding what the software needs to protect against and makes it possible to test whether these requirements are fulfilled at a later stage in the cycle.

4. **Secure Application Design.** Performed in the Design subphase. By carefully examining the design and modelling all threats that could try to attack the software a better understanding of the risks can be achieved. And only once the risks are understood it is possible to take appropriate action, such as removing, reducing, transferring or accepting the risk.

5. **Secure Coding.** Performed in the Development subphase. Secure coding refers to writing application code securely to prevent security bugs from being implemented. Ideally the organisation has set up in advance secure coding guidelines that all developers have to adhere to. Avoiding unsafe language constructs, validating all input and output, properly handling exceptions and performing adequate logging are some of the many aspects that are important during this activity.

6. **Application Security Testing.** Performed in the Testing subphase. In addition to functional and acceptance testing, it is important to also perform security testing to verify the security of the software. In this activity the software is tested whether it implements all defined security requirements, but also if it does not contain other common security issues.

7. **Application Security Configuration.** Performed in the Deployment subphase. During this activity the production system is hardened to ensure that the security features implemented in the software cannot be bypassed via the underlying platform. This activity includes disabling unnecessary services and programs on the production system, installing all patches, removing default (administrative) accounts.

8. **Application Security Verification.** Performed in the Operations & Maintenance phase. Periodically the security of the application should be verified, to check for new security vulnerabilities that may have been introduced due to maintenance or enhancements. The frequency of the verification activity should depend both on the application’s security level and the frequency of updates on the released product.
Figure 2.4: The OWASP methodology to integrate security into the systems development life cycle [27].
Chapter 3

Verifying Software Security

In the previous chapter we defined information security and software security and showed how to secure the systems development life cycle in order to systematically develop secure software. In this chapter we look into how we can verify the security of software once it has been developed.

In the first section we investigate different, established methods to verify the security of software. In the second section we dive deeper into the world of one of these ways: the security code review. We describe three different approaches to software security and identify the overlap to come to a generic approach. At the end of the chapter we look into the advantages and disadvantages of security code reviews and whether the disadvantages can be mitigated somewhat by combining security code review with a different software security verification method.

3.1 Software Security Verification Methods

The methodology to verify software security has already been discussed in the introduction: the Information Systems audit. Since software is an IS, it can be audited to verify the security. However, this may not be as easy as it sounds. Since software security is an idea and not something tangible, it is difficult to measure it quantitatively. In addition, also a qualitative analysis can be difficult, since some security vulnerabilities in software can require precise or rare circumstances to actually surface (e.g. a race condition\footnote{https://www.owasp.org/index.php/Race_Conditions}).

Nonetheless, a few different types of IS audits to verify software security have been developed. Below some of the most common methods are described.

3.1.1 Software Development Process Review

All software has been developed at some point in time. Software could be automatically generated (for example using Model-Driven Development \[26\]), but currently most software is still developed by human beings. In the previous chapter we introduced the systems development life cycle and how security can be incorporated in this life cycle. An independent auditor can verify the security...
controls implemented in the systems development life cycle and thereby give an opinion on the effectiveness on security in the systems development life cycle.

The auditor can verify both the design and the effectiveness of the development process, for example by studying the process description, performing sampling on developed software and interviewing developers. This can give reasonable assurance on the effectiveness of the software development process. The idea is that an effective software development process leads to more secure software, but this is not necessarily the case. Nonetheless, an effective software development process is more likely to result in software security being built-in in a standardised and consistent way in all software developed by the organisation.

3.1.2 Software Certification

Besides the review of the software development process it is possible to certify specific software. Different control frameworks can be used by an IS auditor to verify whether the software meets established criteria for certification. Some of the available standard control frameworks are listed below:

- **Common Criteria for Information Technology Security Evaluation** [10]. The Common Criteria (CC) is a framework through which the assurance regarding the specification, implementation and evaluation of a computer security product can be obtained in a standardised and repeatable way. The Common Criteria define a standardised way to select security and assurance functions to which a Target Of Evaluation (TOE) should conform. The functions can be based on a Protection Profile (PP), i.e. a standard set of security functions for certain classes of systems (e.g. an Operating System or a firewall). All functions to which the TOE conforms, as well as assurance measures in place to assure compliance to these security functions, are documented in advance. After implementation the compliance of the TOE to its defined security and assurance functions can be evaluated in different ways. The evaluation results in an Evaluation Assurance Level (EAL) score ranging from 1 to 7, where 1 denotes only functional testing has been performed on the final product, while 7 denotes that the complete design and all testing has been formally verified.

Even though Common Criteria standardises the assurance of security, the resulting EAL does not say anything by itself. Any user should carefully examine the specified security and assurance functions, as well as the Protection Profile, to be able to interpret the EAL or to compare the Common Criteria results of different TOEs.

- **OWASP Application Security Verification Standard** [38]. The Open Web Application Security Project will be known by most for the OWASP Top 10, a list of the ten most prevalent web application weaknesses published every three years. Besides the Top 10 the OWASP publishes a lot of documents and guides that can help developing secure software or testing the security of software. One of these documents is the Application Security Verification Standard (ASVS). The ASVS defines three different verification levels and a large set of requirements that can be verified in web applications. Some requirements are only required for the higher verification levels, i.e. the most critical applications. In addition the ASVS gives
industry-specific guidance for different types of applications and to which ASVS level the OWASP suggests that the application conforms.

• **NCSC ICT Security Guidelines for Web Applications** [29] (only in Dutch). The Dutch National Cyber Security Centre has published a white paper with security guidelines for web applications. It contains a very large and extensive set of control objectives that a secure web application should conform to, ranging from tactical objectives within the entire organisation (such as “Information security is designed as a process”) to very specific technical objectives (such as “Normalise all user input before validation”). Of course a risk analysis should be performed for each web application to prevent implementing control objectives that cost more than the damage of a potential security breach. Compliance to a pre-defined subset of the control objectives from this white paper is mandatory for all websites that make use of the Dutch online identity DigiD in order to reduce the risks of fraud or misuse of these online identities.[2]

It is worth noting that there is also an ISO standard that describes quality criteria for software and this standard includes a section on security specifically: ISO 25010:2011, the successor of ISO 9126-1:2001 [18, 17]. Unlike many other ISO standards this standard only defines criteria and does not contain any specific control objectives. Certification against this standard is therefore not possible, it should be used during the software development process to ensure that all software quality criteria are covered in the development process and therefore are present in the resulting software.

### 3.1.3 Static Analysis

Another method to verify software security is static analysis. Static analysis is the (automated) analysis of source code or the compiled binary without actually executing it [7]. A static analyser generally does the first few steps a compiler does as well, such as tokenising and parsing the source code. Then the analyser attempts to find dangerous programming constructs or programming errors. Many different static analysers exist, some are designed to find specific types of programming errors in specific languages, while others are more generic and can handle many different errors in many languages.

### 3.1.4 Dynamic Analysis

Dynamic analysis is often contrasted with static analysis. In dynamic analysis the behaviour of the software is analysed while it is being executed [37]. Typically it involves automated monitoring of the execution of different execution paths through the program. In order to reach the different execution paths the dynamic analyser requires a set of test cases, which should be designed in such a way to cover as many execution paths as possible.

### 3.1.5 Fuzz testing

During fuzz testing or fuzzing a very large number of invalid or erroneous input data is generated and sent to the software while hoping to trigger some unex-

pected or strange behaviour. Anything out of the ordinary may point at a programming error, which may be an exploitable security vulnerability. Different fuzzing techniques exist, which technique can be used often depends on the knowledge of the software under scrutiny and the available time. The following three different techniques exist:

1. **Black-box or ‘dumb’ fuzzing.** A technique to find technical security errors by generating completely random input the software. This method can be applied to any program that accepts input (this can be user provided input, but also a file, a database, etc.)

2. **Protocol or ‘smart’ fuzzing.** A technique to find technical security errors in software that implements a specific protocol (for example TCP or HTTP). With this approach the fuzzer does not generate completely random input, but instead uses the protocol specification to generate more clever random input, such as corner cases and incorrect values for length fields. Because the input often does adhere to some parts of the protocol, the input is more likely to hit different parts of the software flow. On the other hand, the fuzzer has to be adapted for each protocol specifically, therefore being more labour intensive.

3. **State fuzzing.** A technique to find errors in the internal state machine of the software, as opposed to finding technical security errors. A state fuzzer sends valid (or almost valid) input to the software, but does so in the wrong order. If the software accepts certain input at the wrong time, the consequences could be dire. For example, if the software implements a sale process and the payment step can be skipped, the organisations using the software can lose a lot of money on unpaid goods.

### 3.1.6 Penetration Testing

In a penetration test the tester attempts to simulate a hacker and uses the same tricks, tactics and techniques as a real hacker would. By thinking and acting like a malicious hacker vulnerabilities can be identified. Automated vulnerability scanning tools help the penetration tester to identify low-hanging fruit or to test all input fields for common input validation errors. But besides the automated scanner the penetration tester has to manually test many aspects of the software (e.g. authentication and session management) to find more subtle and very specific vulnerabilities.

### 3.1.7 Security Code Review

And of course, last but not least: security code review. This verification method is the main topic of this thesis and will therefore be described in more detail in the next section.

### 3.2 Security Code Review

In the introduction we already introduced the security code review as a subset of a source code review. In a source code review the reviewer attempts to identify
any potential issue within software by inspecting the source code. These issues could be potential security vulnerabilities, but may also be other (potential) problems such as sub-optimal performance or high code complexity. The latter two do not necessarily compromise to the security of the program, but are likely to increase the costs of running the software or performing maintenance on the software respectively.

In a security code review the reviewer is only looking for potential security vulnerabilities. Therefore the reviewer focuses his efforts on those parts of the program that constitute the highest security risks, in order to reduce the time required for the review while still being able to guarantee high quality.

As already established in the introduction different security code review process descriptions exist. We will describe these three different processes in more detail and then identify the common steps.

3.2.1 OWASP Security Code Review Process

OWASP provides an extensive description for the security review process online. Unfortunately they do not provide a clear overview of the steps to take in the review, but rather give an assortment of activities to perform. We have summarised these activities below:

1. **Understanding the context.** The first step is to understand the application and the context in which the application runs. What is the business purpose of the application? And what business can be impacted by abuse of this application? By modelling the threats to the application the key risks can be identified and the review can focus on the mitigations implemented for these key risks. This can be done by studying requirements and design documentation and interviewing both architects, developers, as well as business users. If interviews take too much time, the same results can be achieved using a questionnaire.

2. **Understanding the attack surface.** Once the reviewer understands the context the reviewer should analyse the attack surface. All locations where the application accepts input are potential areas where bugs can result in security vulnerabilities. Using dynamic and static flow analysis tools the reviewer can automate this work partially. Besides data flow analysis the reviewer should also investigate transaction flows. Software performs certain operations on the data and may invoke security relevant functions during this operation. The way the security relevant functions are implemented and invoked (or not invoked) can also be a source of security vulnerabilities.

3. **Verify the applicable security controls.** OWASP provides a checklist with controls that the reviewer can use during the review. Not all security controls will be necessary for every application, therefore the reviewer has to use the output of the first two steps to select those controls that are relevant for the application to be reviewed. The reviewer has to manually check whether all these security controls are in place and implemented correctly.

4. **Rank the threats.** For all identified vulnerabilities the reviewer should rate the risk of this vulnerability. Different methodologies can be used to rate
the risk, but often the risk rating is expressed using the following formula (e.g. in [4, 11, 2]).

\[ \text{Risk} = \text{Probability} \times \text{Impact} \]

By rating the risks it is easier to determine which risks should be mitigated first and if risks should be mitigated at all.

3.2.2 Security Innovations Security Code Review Process

The security code review process created by Security Innovations is a five-step approach with multiple passes over the code [14]. The approach is based on the review process created by Microsoft and published back in 2005 [24]. The approach is summarised below:

1. Identify security code review objectives. In the first phase the goals and the constraints of the review are defined. The constraints include time and scope of the review, as well as the type of security issues that will be reviewed. The objectives for the review include which exact security issues will be reviewed and which threats will be considered during the review.

2. Perform preliminary scan. In the second phase a first scan of the code is made to find initial security issues and hot spots where a lot of security functionality is implemented. This phase can be performed with automated scans, manual scans or both, depending on time and resources available and of course the objectives set in phase one. Hot spots can be found by looking for functions where input/output are handled, where authentication and authorisation is performed and where errors are handled.

3. Review code for security issues. In this phase the reviewer looks for common, generic security vulnerabilities in the code. By using the output of phase two the reviewer can focus efforts on those parts of the code that are the most prone to security issues. Microsoft recommends using a question-driven approach combined with control flow and data flow analysis. The question-driven approach is very similar to the standard audit approach with a list of control objectives, except that the objectives are formulated as questions. Control flow analysis refers to analysing how the software steps through the code, including branches, loops and error handling. Data flow analysis refers to tracing how input is transformed to output and how potentially malicious input is validated.

4. Review for security issues unique to the architecture. Where the reviewer looks for generic security issues in phase three, the reviewer looks for software specific issues in phase four. This phase includes looking at custom implementations of security features (such as encryption), any implemented risk mitigation techniques and verification of privileges in case different authorisation roles exist in the application. The reason that this phase is separated from the common security vulnerabilities, is that any bug in these specific parts is very likely to result in a security vulnerability. Therefore these parts require special attention from the reviewer.
5. **Prioritise and remediate the findings.** Once the review has been performed, the reviewer should prioritise the findings with regards to the business risk it poses, especially for the customers. Since this document is aimed at organisations reviewing their own software, the next step is to fix the bugs and learn from your mistakes. For an independent auditor it is not possible to fix the bugs, but the auditor can give recommendations on how to fix the bugs.

### 3.2.3 Infosec Institute Security Code Review Process

The Infosec Institute is another organisation that has published a security code review process [31]. It is not as elaborate as those from OWASP and Microsoft, but does contain the most important steps in the process. Below we give a short summary of the approach.

1. **Define scope.** In the first step the scope of the review should be defined. Questions such as time and resource constraints, which code will be reviewed and which types of vulnerabilities will be reviewed should be answered in this step.

2. **Interview developers.** By talking to the developers the reviewer can get a better understanding of the software. In addition, it can help to give the developers a good feeling that you are trying to help them instead of only pointing out mistakes they make.

3. **Categorise.** During the review focus at first on the types of vulnerabilities that are most likely to have a high risk. This could be in the business logic of business applications, or in specific technical areas such as authentication, authorisations or cryptography.

4. **Give recommendations.** Each potential vulnerability should be verified, especially those originating from tools as they could be false-positives. For each of the actual vulnerabilities a recommendation to remove the cause should be given. The developers can use this recommendation as the solution or at least as a direction for the solution.

### 3.2.4 Generic Security Code Review Process

In the previous three subsections we looked at three different security code review processes. Although these processes do show differences, they all boil down to the following multi-step approach:

1. define the scope and objectives;
2. perform a risk analysis;
3. define the security control objectives that will be verified;
4. verify the security control objectives;
5. report the results and recommendations.

Each of these steps will be further detailed and explained in the remainder of this section.
1. Define objectives and scope. Just as about any other process one should start with a good preparation for the activities. Therefore the very first step should be identifying the objectives and scope of the security code review. Is the review to verify compliance to certain given standards? Is it intended to find as many potential vulnerabilities as possible? Should it only focus on specific vulnerabilities? Is the entire software code base in scope or only specific parts? The objectives for the review define to a large degree how the remainder of the preparation and review will be performed. All further activities should always keep the objectives in mind and if they do not help to complete the objectives they should be skipped.

Objective identification typically has to be performed by means of an interview or workshop with the security code review sponsor(s).

2. Perform a risk analysis. Once the objectives have been identified the reviewer can start the risk analysis. By investigating different aspects of the software the reviewer can identify what the main security risks are for the software and where therefore most of the effort should be focussed on in the later stages of the security code reviewing process. This helps to reduce the time and costs required for the review, while only having a limited impact on the quality of the review.

The risk analysis is typically performed by interviewing the software product owner, software architects and designers and studying available documentation.

3. Define the security control objectives. Once the objectives and scope are clear and the risks to the software have been identified it is time to define the security control objectives against which the software will be verified. Standards and best practices, such as from OWASP, Microsoft or the NCSC \[38, 24, 29\] can form a solid basis for this step. The basis should then be extended with application specific risks identified in the risk analysis. In addition, controls objectives that do not fall under the review objectives should be removed.

This step is typically performed by finding the right basis and using professional experience and judgement to formulate the right security control objectives. The final list should be reviewed and approved by the client.

4. Verify the security control objectives. With the list of security control objectives in hand the reviewer can start the actual review. This process will take the majority of the time of the entire review, as it involves studying the source code to verify all security control objectives. Automated tools can help improve this process, for example by looking for standard security vulnerabilities.

The exact activities in this step are dependent on the security control objectives. However, there are two activities that almost always have to be performed. The first is to identify a large class of technical vulnerabilities, while the second is to identify different logical vulnerabilities.

- Plot and analyse data flows. Identify where data enters and where data exits the software. Anywhere data enters or exits the software
potentially exploitable security vulnerabilities can exist. If the software does not allow for any user input, then any vulnerabilities will not be exploitable by the outside attacker. Only when the attacker is able to manipulate the data that the software processes the attacker may be able to exploit existing vulnerabilities.

Once the reviewer has identified the entry and exit points of data in the software, the reviewer can plot how the data flows from the entry points to the exit points. These data flows can then be analysed to ensure that the appropriate input validation and (if required) output encoding techniques are implemented in the software. Inadequate input validation often leads to exploitable security vulnerabilities.

Ideally all data flows in the software form an hourglass shape as depicted in figure 3.1. By implementing all input validation and output encoding in a single module and ensuring that all data flows pass through this validation module, the validation process as a whole is much simpler and easier to understand for both the developer and the reviewer. As a result the developer is less likely to make mistakes in the input validation process or forget validation for a certain entry point. For the reviewer the hourglass shape means effort can be focussed on the central validation routines and verifying that all data flows conform to this hourglass shape.

Using data flow analysis the reviewer is able to investigate a large number of different classes of technical vulnerabilities at once. These classes include, but are not limited to, buffer overflows, format string vulnerabilities, injection (e.g. SQL, OS, command, log), Cross-Site Scripting, arbitrary file access, inadequate verification of authorisations and insecure storage of sensitive data.

This phase of the review is easiest if there is clear architecture documentation available that models the data flows. The correctness and completeness of the documentation can be verified by the reviewer by inspecting (parts of) the source code. When there is limited documentation available the reviewer has to inspect the source code more extensively. Optionally the reviewer can use automated tools to as-

Figure 3.1: Ideally data flows within software form an hourglass shape.
Identify and analyse transactions. While data flow analysis is very suitable for finding technical vulnerabilities, it does not help the reviewer in detecting vulnerabilities in the logical program flow. Most programs do some processing on input data before it is sent to the output channels. Often this data processing process is a multi-step process where the steps have to be performed in a specific order. Being able to break the expected data processing flow by skipping certain steps may introduce exploitable vulnerabilities in the software.

Take for example the process flow of a purchase transaction in a simple web shop depicted in Figure 3.2. The customer first has to select an item, then pay and finally the customer is congratulated with their purchase and the selected item is sent to the customer. The arrows in Figure 3.2 depict the expected logical flow. For example, the customer is allowed to cancel a selected item before paying, but is not allowed to go to the congratulations page without paying. Now if the web shop software does not correctly force this logical flow, but instead allows the customer to bypass the paying step, this could lead to unpaid items being delivered to the “customer”.

In order to detect vulnerabilities in the logical flow the reviewer has to identify and analyse all transactions that occur in the software. For each transaction the reviewer has to verify that the steps in the transaction can only be performed in the expected order. Ideally a standardised way to perform these checks is implemented. Many standardised ways to implement logical flow checks exist, but similar to using the hourglass shape for data flows choosing one standardised approach within a piece of software increases the understanding for both the developer and the reviewer.

Using transaction flow analysis several different types of logical flow vulnerabilities can be detected, such as inadequate protection of sensitive data (lack of authentication), inadequate verification of authorisations, inadequate session management and Cross-Site Request Forgery.

Unfortunately for the reviewer very few tools exist to assist with the transaction flow analysis process. This analysis requires reasoning about the software and the implemented processes, something automated tools are typically very bad at. As a result the reviewer has
to perform a manual inspection of the code to identify these types of flaws. Interviews with developers/designers and reviewing documentation help the reviewer to understand the expected process flows and allows them to reason about them.

5. **Report the results.** Once the reviewer has completed the verification of all security control objectives the reviewer has to document the results. This last phase consists of three steps, as described below.

   (a) **Rate the identified risks.** During the review the reviewer may have identified vulnerabilities in the software. For each of these vulnerabilities the reviewer should determine the risk rating. This can be done using the standard Risk = Probability × Impact formula introduced before, or with a more elaborate risk rating scheme.

   The risk rating gives the reader of the report an idea on the prioritisation to mitigate the identified risks. The vulnerabilities with the highest risk ratings are likely to occur, have a high impact or both and should generally be mitigated before vulnerabilities with a low risk rating.

   Determining the exploit probability for a vulnerability is often relatively easy, as the reviewer can estimate how much interaction is required and how many implemented security measures have to be circumvented for successful exploitation. The business impact is usually more difficult to estimate, as it requires the reviewer to know what the assets of the organisation are and how much it will affect the business if these assets are (partially) compromised.

   (b) **Formulate potential solutions.** In order to help the organisation as much as possible the reviewer can formulate potential solutions to fix the identified vulnerabilities. Sometimes these solutions can be trivial (e.g. removing some debug statements), while others could require extensive work (e.g. for flaws in the security architecture or design). The potential solutions can also help the organisation in prioritising the order of fixing, as it allows them to determine the low-hanging fruit (i.e. high risk, simple fix vulnerabilities).

   (c) **Write the report.** Once the reviewer has finished the review and determined the risk ratings and potential solutions the reviewer can write the report. The report is used to formally document the results and optionally to give an overall verdict. The exact format of the report is up to the reviewer and the receiving organisation, but usually the report contains at least an introduction (including objectives and scope descriptions), a non-technical management summary (for higher management) and a list of all security control objectives and whether the software complies to these objectives or not (including evidence). The reviewer should first supply a draft report to the organisation, so that they can provide their opinion on the findings, risk ratings and potential solutions. After approval the final report can be issued.
Table 3.1: Relative costs to repair defects when found at different stages of the life cycle, taken from [39].

<table>
<thead>
<tr>
<th>Life cycle stage</th>
<th>Baziuik (1995) study costs to repair when found</th>
<th>Boehm (1976) study costs to repair when found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>1X(^b)</td>
<td>0.2Y</td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td>0.5Y</td>
</tr>
<tr>
<td>Coding</td>
<td></td>
<td>1.2Y</td>
</tr>
<tr>
<td>Unit Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Testing</td>
<td>90X</td>
<td>5Y</td>
</tr>
<tr>
<td>Installation Testing</td>
<td>90X-440X</td>
<td>15Y</td>
</tr>
<tr>
<td>Acceptance Testing</td>
<td>440X</td>
<td></td>
</tr>
<tr>
<td>Operation and Maintenance</td>
<td>470X-880X(^c)</td>
<td></td>
</tr>
</tbody>
</table>

\(^a\) Assuming cost of repair during requirements is approximately equivalent to cost of repair during analysis in the Boehm (1976) study [3].

\(^b\) Assuming cost to repair during requirements is approximately equivalent to cost of an hardware line card return in Baziuik (1995) study [1].

\(^c\) Possibly as high as 2900X if an engineering change order is required.

### 3.3 Advantages and Disadvantages of Security Code Reviews

In this section we will look into the advantages and disadvantages of security code reviews. These will most likely be considered when security code reviews are used or not used in practice.

#### 3.3.1 Security Code Review Advantages

Security code reviews have one huge advantage over other software security verification methodologies: it can cover every single line of code and therefore every single aspect of the software. Using a security code review it is potentially possible to identify every single flaw in the software. This is something no other verification method is able to do. As a result of being able to reach full coverage, utilising security code reviews it is possible to give reasonable assurance over the security of software, as already pointed out in the introduction.

Security code reviews can be performed early in the systems development life cycle, as opposed to several other methods such as binary analysis or penetration testing. Although the exact numbers vary greatly, it has been generally accepted that fixing bugs early in the development cycle is (much) cheaper than fixing bugs later in the cycle, especially when the software has already been taken in production. Two studies that have shown this are compared in table 3.1 Therefore applying security code reviews early can save costs and effort to fix bugs, as well as potential reputational damages that could occur when a bug is found in production [31].
In addition, security code reviews are very suitable for finding instances of “security by obscurity”, i.e. features that are obscured with the idea that they are protected from misuse. For example the software may have a hard-coded but hidden administrator account that can serve as a backdoor into the application. As long as this backdoor is protected using a strong password it is not likely, but also not impossible, that it will be found when the software is used normally or under attack by malicious users. With access to the source code the reviewer can easily find such backdoors.

### 3.3.2 Security Code Review Disadvantages

Security code reviews have several disadvantages that might make it less suitable for a specific software security verification. The first of the disadvantages is also one of the most commonly heard: it is slow and therefore expensive. Even though in theory security code review could identify every single flaw in software, this would often be too expensive. The majority of the security code review has to be performed manually, which increases the costs even more [8].

Another disadvantage of security code reviews is that it depends heavily on human factors. The quality of the review depends on the skills and experience with the used technologies, architectures and scenarios of the reviewer. Also the reviewer is susceptible to human error, fatigue or boredom that can lower the quality of the review [8] [12].

A third disadvantage has to do with coverage of the execution environment. The software will eventually be executed on an IS infrastructure, such as an Operating System and maybe also middleware. In section 2.3.2 step 7 we described how securing the execution environment should be part of the systems development life cycle according to OWASP [27]. Unfortunately, security code reviews do not extend to the infrastructure and therefore cannot be used to verify the security of the execution environment. This has to be verified using a different approach.

Another disadvantage of security code reviews is that it is often not possible to verify whether an identified vulnerability is really exploitable in the software. The reviewer may come across potential vulnerabilities during the review, but it is possible that somewhere else in the code mitigating measures are implemented. Due to time constraints the reviewer is often not able to check every line to look for counter measures. In addition, certain counter measures may be implemented in the execution environment, which is generally not part of the review as discussed above. The result is that security code reviews often lead to a list of “potential vulnerabilities”, which may or may not be exploitable in the software. For the organisation receiving the results therefore a lot of effort is required to verify the potential findings and test whether they are actual vulnerabilities.

Lastly a disadvantage for security code reviews is that it requires access to the source code. This has several implications. At first, if the application is developed by a third party, access to the source code may not be possible. This may also be the case for source code of third party libraries that are used in the software. In addition, organisations may be hesitant to give the reviewer access to the source code, as they are afraid their intellectual property might leak out in this process. Contractual agreements can alleviate the latter concern, but they will not take it away completely.
Chapter 4

Expert Opinions on Security Code Reviews in Practice

In the previous two chapters we defined information security, software security and how security ties into the systems development life cycle. We have also described different methods to verify software security, with emphasis on the security code review. We have summarised three different approaches and defined a generic approach to execute a security code review. Lastly we have discussed several advantages and disadvantages of security code reviews, both with relation to software security and with relation to other verification methods. This has answered the first two research subquestions and concludes the theoretical study of this thesis.

In the next two chapters we will investigate security code reviews in practice and determine which considerations there are for using security code reviews. We will start of in this chapter with the opinions of several subject matter experts. We explain whom we interviewed, discover why security code reviews are not (yet) used often in practice and whether this is a good or a bad thing. In the next chapter we look into the software risk assessment process at a Dutch bank. We describe the considerations that is used in this process to determine whether software verification should be through security code reviews or some other method.

4.1 Interviewees

For this practical study we interviewed five different persons with three different roles and functions at different organisations:

1. An IT Security Consultant and IS Auditor, who is manager at a large audit and advisory firm. He manages engagements to help his clients better secure their information assets. He has performed several engagements that included a security code review.

2. An IT Security Consultant, who is also manager at a large audit and
advisory firm. In the previous years he has performed many source code
and security code reviews for clients.

3. Three Risk Analysts for a Dutch bank, who are responsible for assessing
the risks for different applications developed for the bank or its daughter
firms, including business critical applications such as Internet Banking and
applications related to the Public Key Infrastructure. The three Risk An-
yalysts have different applications focuses, such as Internet-facing or Pub-
lic Key Infrastructure. For all changes to these products they determine
whether the security of the changes has to be verified and which methods
are used to perform this verification (security code review, penetration
testing, configuration review, etc. or a combination of multiple methods).
After the verification they also help the product owner to understand the
(technical) security risks, so they can make an informed decision whether
to mitigate or accept the risks.

4.2 Subject Matter Expert Opinions

In this section we give background information we got from the interviews.

4.2.1 “The threat landscape has changed”

One of the reasons that were mentioned why security code review are not yet
used often has to do with the threat landscape of software and information
systems. About a decade ago the threat was mostly outsiders that worked on
their own. Therefore for organisations it was often sufficient to know whether
one of these threat actors could enter their systems within a specific time frame,
something typically covered by a ‘black box’ penetration test. In such a test
the penetration tester has no prior knowledge about the information system
and software and attempts to gain unauthorised access or otherwise hamper the
functioning of the system.

However, the threat landscape has changed significantly. The same threat
actors still exist, but are overshadowed by cybercriminals that work in organ-
ised gangs or even in name of a nation. Even though the numbers that describe
the economic impact of cybercrime and cyber espionage vary wildly ([20]), it
is clear that it is serious business. In order to defend against these organised
actors organisations want to know all their vulnerabilities. Security code re-
views can help organisations to get closer to this goal. However, because many
organisations are still struggling with their basic information system security,
the use of security code reviews to prevent more advanced threats is not yet
widespread.

4.2.2 “Secure by Design”

All interviewees were very clear about one thing: it all starts with a systems
development life cycle in which security is integrated. Security requirements
for software should be defined in advance and the resulting software should be
tested for these requirements before it leaves the development phase. This forms
the basis for software that is secure by design. This provides a few advantages:
• Several potential security issues are never introduced into the software, hence do not have to be fixed.

• Other potential security issues that are introduced, can be found and fixed in the development phase, which is significantly cheaper.

Unfortunately organisations do not always have full control over the development life cycle, since the software may be developed at a third party. In this case the organisation has to make contractual agreements with the third party regarding the security requirements for the software. Upon delivery of the software the organisation can test for these requirements and in case of malversations take it up with the developing party. Other measures to compensate for the lack of control over the development life cycle of a third party is to request and evaluate internal (security) test results of the third party or request an independent IT auditor to audit the development life cycle.

4.2.3 “The goal is ‘secure enough’”

Another point all five interviewees raised is that the goal of software security verification for an organisation is to gain sufficient comfort that the software is secure according to their standards. Organisations should define for themselves what their software security standards are to which software should comply. This can of course be an official standard or criteria for certification, but this is not necessarily the case. If the software verification process then shows that it complies with the standards, the software is deemed ‘secure enough’ for the organisation. This can be viewed as an certification, even when it may not comply with an official, public standard.
Chapter 5

Security Code Review
Considerations in Practice

The Risk Analysts determine for each new application or change to an existing application (from now on simply called ‘change’) what the impact of the change is with regard to the risks for the bank. If the change has an impact on the risks for the bank, then the security of the software that was changed has to be verified. This can either be a verification of the entire application, or of only a portion (in case only certain functions of an existing application were changes). Which methods are used for the verification are determined on a case-by-case basis, with several considerations in mind. In this section we describe the considerations, both the positive and the negative.

Please note that most of the software development, as well as the actual security verification process of changes is not performed by employees of the bank or its daughter firms. Instead, trusted third parties develop most of the software, while other trusted third parties verify the security. These third parties have been carefully selected and have a contractual agreement with the bank, which stipulates several rules of engagement.

5.1 Risk Analysts Roles in Projects

Projects within this Dutch bank are managed using the PRINCE2 methodology. In this subsection we describe which roles the Risk Analysts play during projects and how they help the business in understanding their risks.

Ideally the Risk Analysts are contacted before a project is even started (to be more precise, when the PRINCE2 Project Mandate has been created). All project ideas are discussed in a special risk assessment board. This board consists not only of Risk Analysts, but also of employees from other departments involved in risks, such as Legal, Compliance and Internal Audit. The board performs a high level risk assessment in order to identify the risks that this project poses for the bank. These risks are generically formulated and not specifically targeted at software or software security.

When the project is started up after this meeting, the Risk Analysts perform a technical risk analysis. This analysis is focussed on the technical implications of the project, such as changes to application. The Risk Analysts do not only
identify the technical risks in this step, but also formulate security requirements for the project, so the project team can limit the risks in an early stage in the project.

During the project the Risk Analysts are always available for questions from the project team. This is a consulting role, which means that the project team has to initiate communication. It is not required to do so, but of course it is better for all parties if it does.

Then before the project is finalised and the changes go live, the Risk Analysts write a risk report. This report contains of course the security requirements, but also whether these requirements or other mitigating measures are implemented. In addition, it contains an analysis of the application risks against the risk requirements of the bank, as well as recommendations for the project team. This report is handed over to the project team and a required step for each project. The project team decides, together with the business, what to do these risks and recommendations. They decide which risks are mitigated, transferred or accepted. In case risks are mitigated, the Risk Analysts are once again asked, but this time to verify to what extent the mitigating measures actually lower the risks. The business can then decide whether these measures lower the risks to acceptable levels.

Of course, not every single project follows this ideal process. Some projects have a very short turnaround time of only a handful of weeks, which makes it impossible to go through all of these steps. In this case, a slimmed down version of this process is followed. The risk assessment board is then skipped, as is often the definition of the technical security requirements by the Risk Analysts at the start and the consulting role of the Risk Analysts. What is explicitly not skipped, is the risk analysis report. When there is limited time the Risk Analysts determine the security requirements and in parallel the security of the software is verified by an external party. The results are combined in the risk analysis report. This means that the software is verified against security requirements that were not known before the software was built, which could result in serious security risks. It is then up to the business to determine whether these risks are acceptable for them or not.

5.2 Reasons to Perform Security Code Review

The Risk Analysts of the bank have been employing security code reviews as a means to assess software security for many years. Of course they have their reasons for doing so. Exactly these reasons are described in this subsection.

- The bank wants to have insight in all its risks, whether they stem from business processes or software. The Risk Analysts are there to assess the risks from software. Therefore, the Risk Analysts have as goal to learn as many vulnerabilities in the software as possible. Security code reviews are a good means to reach this goal, for reasons already described in this thesis, such as the high coverage rate and ability to find backdoors. Security code reviews are rarely used as only means of verifying the security, but also in combination with other methods, they show their value time and time again.

- For most third parties that carry out the software security verification
penetration testing is their core competency. Therefore penetration testing is the main verification method that is employed for almost all changes. According to the Risk Analysts (as well as the other two subject matter experts) security code reviews can have great synergy with penetration testing when performed hand in hand. Being able to look under the hood of the application while penetration testing increases the effectiveness of the penetration test and therefore the software security verification.

For example, when certain unexpected behaviour is found with the penetration test, the source code can be used to determine what causes it and whether this is an actual security vulnerability that can be exploited or just a functional bug. Without the code the penetration tester might not have the time to find out if this is a real vulnerability. In addition, the tester is not able to report the cause of this bug, which increases the effort to fix it.

Another example of the synergy between penetration testing and security code reviews works the other way around. During the security code review it is possible to identify complex pieces of code or pieces of code that implement major security functionality (e.g. input validation or authentication). Testing hands-on whether this functionality works as expected increases the understanding of the code, as well as helps preventing false-positive findings.

- Security code reviews do not greatly increase costs for the verification of a lot of applications, especially when combined with penetration testing. This gives the bank the best of both worlds: higher coverage without spending more money. For applications where security code reviews may have high costs (e.g. when the code is very complex or written in an uncommon programming language), the Risk Analysts often choose for other methods to assess the security.

5.3 Risk Analysis Process

The benefits of security code reviews are clear to the Risk Analysts, yet still they do not always select this software security verification method. In this subsection we look at the risk analysis process that the Risk Analysts follow at the bank.

The risk analysis process is not completely set in stone. Several guidelines and standard ways of working exist for the Risk Analysts, but in the end the judgement and professional experience of the Risk Analyst also an important role.

One of the standards within the bank says, that all applications that are available externally (i.e. on the Internet) have to be subjected to a penetration test. For applications that process personally identifiable information (PII) or financial account details a pentest is also almost always required. Banking applications are furthermore often subjected to a security code review, but this is not a very strong requirement.

Besides these bank-wide standards, each of the different Risk Analysts also have some of their own standards to which they abide. This is probably sparked
by both the application focus they have within the bank, as well as the expe-
rience they have with the different verification methods. For example, one of
the Risk Analysts prefers to have authentication modules both subjected to a
penetration test and a security code review, while the other Risk Analysts prefer
to judge this on a case-by-case basis.

5.4 Considerations For Verifying Security with
Security Code Reviews

In this section we describe the considerations that are the most important for
the Risk Analysts to determine how software security is verified and whether
a security code review is performed as part of this verification. All these con-
siderations together form a risk profile, as well as some preconditions, for the
application.

This information is combined by the Risk Analysts to determine which soft-
ware security verification methods are employed. Of course the bank standards
discussed in the previous section are implemented, but this is not always enough.
This is mostly an economic consideration: a comparison between the financial
risks for the bank in case certain potential security risks are not identified versus
the costs of finding and fixing these potential security risks.

Below the considerations of the Risk Analysts are described.

- Probably the most important consideration is the risk rating of the ap-
  plication and the data it processes. The confidentiality, integrity and
  availability requirements of the application and the processes data have
to be rated separately by the business on three point scale. These ratings
are used by the Risk Analysts to determine how critical security is for
this application. In addition, the Risk Analysts needs to know whether
personally identifiable information is process by the application, which
increases the risk profile of the application.

- Another important aspect of the software is whether it is internally (i.e.
intranet) or externally (i.e. Internet) facing. Applications that are exter-
nally facing are exposed to more threats and are more likely to be the
target of an attack. This increases the risk profile of the application and
may warrant a more thorough verification of the security.

- The third aspect the Risk Analysts consider is the type and purpose of the
  environment. This gives the Risk Analyst an idea about how likely this
application is going to be a target for potential attackers. As an example,
the Internet Banking environment is much more likely to be attacked than
a website set up for a marketing campaign that runs for a couple of weeks.

- The Risk Analysts also consider whether the application is hosted inter-
nally or at a service provider. In case it is hosted at a service provider, the
security of all backend systems is more critical, because the bank has lim-
ited control over the network at this service provider. Therefore externally
hosted applications have an increased risk profile.

- It is also relevant for the Risk Analysts whether the security of the ap-
  plication has already been verified before. Sometimes an application is
only changed and spending a lot of money, time and effort into verifying the entire application while most of it has already been verified before, is a waste. Therefore this is also an important consideration for the Risk Analysts.

• The Risk Analysts also consider which software security verification methods are available and to what extent they are able to cover the application. An example given by a Risk Analyst was the verification for an application in which the archiving function was of critical importance. Even after ten years it was important that the archive was still intact. Testing such a long period with a penetration test is not feasible, therefore another method had to complement the penetration test for this particular application.

• A consideration more specific to security code reviews is whether the source code of the change is available or not. For most of the applications the software is developed by a third party, but the bank has contractual ownership of or at least insight into the code once it is finished. In that case the source is available and a security code review could be performed. However, in some cases the ownership of the code is not transferred or right to inspect is not given, which makes a review of the code impossible. In addition, some software may use closed source third party libraries, for which the source code is also not available. In this case the library cannot be reviewed, but the software itself may be.

• Another aspect considered by the Risk Analysts is the language in which the software is programmed. Most recent applications are programmed in common high-level languages such as Java, but older applications in use by the bank may be programmed in COBOL or Fortran. Finding a security code reviewer for an application in COBOL or Fortran is both very hard and very expensive, due to the fact that these particular languages are not used often anymore. In this case the costs of the security code review almost always outweigh the benefits.

• The complexity and size of the code base are also considered by the Risk Analysts. For a security code reviewer it is important to be able to understand the application and to be able to cover the majority of the source code. If the code base is very complex, or the application is very large the effort for the reviewer to understand the source code can be disproportionately large. This results in higher costs for the security code review (as compared to other methods, such as a penetration test), while the benefit of higher coverage may not be that much anymore due to the large amount of code. The Risk Analysts were unable to give figures what too complex or too large is, but instead they use their professional judgement to determine this for each change.

• The availability of the developers of the software is also important, as this can help the security code reviewers gain a better understanding of the software. Often interviews with the developers, or even just being able to ask some questions, can greatly increase the understanding of the application in a short time frame. And a greater understanding of the application leads to a more effective review, by speeding up the “risk
analysis” phase for the reviewer. So while not strictly required, availability of the developers is considered by the Risk Analysts.

- Just like interviews with the developers, documentation can also help the reviewer to understand the source code. Good architecture and design documents also help during the “risk analysis” phase and help the reviewer to identify the security-relevant parts of the application. Therefore, availability of documentation and the quality of this documentation are also considered by the Risk Analysts. If they had to choose between availability of documentation or developers, they would choose developers over documentation. This gives the reviewer the opportunity to ask more than just about the structure of the code, but also about specific coding constructs or implementation choices.

- As pointed out in the previous subsections, the bank would like to know as many risks as possible. Therefore the Risk Analysts also consider whether the infrastructure on which the application is running is sufficiently covered when a security code review is performed. Often this is tackled by the combination with the penetration test as performed by the verifying party, but this may not necessarily be the case. To summarise, the Risk Analysts will generally not choose security code review as the only verification method, as this does not cover the infrastructure. This will be supplemented by a method to verify the security of the infrastructure, such as a penetration test or technical configuration audit.

- Yet another consideration is the preference of the verifying party. The Risk Analysts ask the verifying party for their recommendation on this application and add this recommendation into the equation on what verification methods should be used. If the verifying party desires a security code review, the Risk Analysts will often attempt to make this possible. On the other hand, if the verifying party has strong arguments that they can offer the bank the level of security they desire without security code review, this may make the Risk Analysts decide against a security code review.

5.5 Less Important Considerations

In the interviews it became clear that several considerations we defined in section 3.3 were not so important for the Risk Analysts. Below we have listed these considerations and why they were not so important for the bank.

- The bank was not very afraid of losing control of their intellectual property. They employ a very extensive selection process for the different software security verifying parties. Once the process is finished, several contractual agreements and secrecy agreements are signed to contractually cover this consideration. In case intellectual property is leaked, this helps covering costs that might come from this loss.

- Even though security code reviews can be used to give positive assurance with regards to the security of software, this is currently not what the bank is looking for. The bank has defined their own control objectives
for the security of software, but negative assurance is sufficient assurance for the bank. Each of the verifying parties are tested during the selection process for their ability to verify these control objectives. This gives the bank sufficient assurance that once one of these parties has verified the security (and the findings are processed by the bank), the application is sufficiently secure.

This may change in the future, mostly because of rules and requirements coming from third parties, such as credit card companies or central banking authorities. For example, in the future it may be required for compliance to PCI-DSS that highly critical software (e.g. software extensions implemented on Hardware Security Modules\(^1\)) is formally certified. This certification process is very costly and time-consuming and not yet required.

- Another advantage of security code reviews that the bank does not consider is the fact that security code reviews can be employed early in the systems development life cycle. Almost all software development for the bank is outsourced to third party providers and covered by contractual agreements. This often includes a fixed fee price for the product, as well as the security control objectives the application should adhere to. It is up to the third party to ensure that their systems development life cycle delivers this level of security, or otherwise they will have to spend a lot of their own time and money to fix any identified security issues. In other words, the bank does not directly suffer from finding the security issues later and therefore does not force the use of security code reviews early in the development life cycle.

\(^1\) A Hardware Security Module or HSM is used to securely store the private keys that are used within an organisation. Access to these keys allows the owner to impersonate the bank to others, such as clients, suppliers or others.
Chapter 6

Conclusion

In this research we discussed security code review as a method to verify the security of software. We looked at how it compares with other software security verification methods, as well as how it could be combined with other methods to gain effectiveness or efficiency. We also interviewed several subject matter experts to get to know more about the use of security code reviews in practice, what the gain is and what the downsides are or may be. In this chapter we will answer the main research question and describe the added value of security code review for software security from an IS audit perspective.

6.1 Answering the Research Questions

In this section we will answer the research questions. We first start with the sub-questions and continue with the main question.

6.1.1 What is software security and which verification methods exist?

In Chapter 2 we defined software security as follows, based on the definition by McGraw:

**Software Security**: The idea of engineering software so that it fulfills functional security requirements set by the user and continues to function correctly under all circumstances.

In Chapter 3 we described the different methods to verify software security:

- software development process review;
- software certification;
- static analysis;
- dynamic analysis;
- fuzz testing;
- penetration testing;
- security code review.
6.1.2 What is security code review and when can it theoretically be applied?

In Section 3.2 we described three different security code review processes and defined the generic security code review process as follows:

1. define the scope and objectives;
2. perform a risk analysis;
3. define the security control objectives that will be verified;
4. verify the security control objectives;
5. report the results and recommendations.

There is only one theoretical limitation to applying security code reviews: the software must of source code. Since this is the case for nearly all software, security code review can be applied on nearly all software.

6.1.3 What are the main elements and characteristics of security code reviews in a practical situation and what are the main values?

Through interviews with subject matter experts we determined when security code reviews are or are not used in practice, as described in chapters 4 and 5. In practice, the main value of security code reviews come from the fact that it is possible to look ‘under the hood’ of the software. Being able to inspect the source code allows the reviewer to identify more risks than what would be possible without the source code.

A major downside of security code review as only verification method is that it is generally more expensive than other methods (e.g. penetration testing). In addition, with a security code review it is only possible to find potential vulnerabilities and it is not possible to verify whether these are actually exploitable in the product. But, combining security code reviews with other methods, such as penetration testing, offsets these downsides and can also improves the effectiveness of the complementing method.

6.1.4 In which situations do security code reviews add value to software security?

This question has been answered by the Risk Analysts. Security code review adds value to software security, if and only if the cost of performing the security code review and fixing the identified defects outweighs the risks of potentially unknown vulnerabilities in the application. Because security code reviews are relatively expensive, this makes them only value-adding for applications that have a higher risk profile. Whether this higher risk profile stems from the processed data, the exposure and type of the application, or the location where it is hosted is not important.
6.1.5 What are the considerations to perform a security code review to verify software security?

Many different aspects have to be considered before a security code review should be used to verify software security. By far the most important consideration is the risk profile of the applications, as described above. Other considerations are:

- goal of the verification (e.g. learning as many risks as possible versus only the critical risks, positive assurance versus negative assurance);
- scope of the verification (e.g. only software or also the underlying infrastructure);
- availability of the source code (in case of external developer or third party products);
- language, complexity and size of the source code (common language or not);
- availability of documentation and the developers;
- protection of intellectual property;
- verifying security early in the development life cycle;
- whether the application has been verified before (e.g. in case of a change to an existing application).

From our interviews with the subject matter experts we learned that some of them were not an issue, such as protection of intellectual property and verifying security early in the development life cycle. The reason is that these particular considerations are covered by contractual agreements with suppliers. Especially larger organisations often have large contracts with different external suppliers, in which these topics are covered, including consequences in case the contractual agreements are not honoured. This makes these considerations less important per application, since they are covered automatically by the large contract.

6.2 Conclusion

Security code reviews are one of the many methods to verify software security. They allow the reviewer to verify many control objectives which cannot be verified with other methods and therefore can be used to give reasonable assurance that a certain set of (security) control objectives is met by the software.

Compared to other software security verification methods security code reviews are often considered slow and expensive. Main reasons are that the reviewer has to get accustomed with the source code structure before the review can even start. Static analysis tools can be used to improve the efficiency and the coverage for large code bases, but manual verification of the results is always necessary. In addition, logical flaws cannot be detected by static analysis tools, since identifying these types of flaws requires understanding of the business process that the software supports. On the other hand, security code reviews can
achieve higher code coverage than any other verification method, which results in a larger number of identified risks in the software.

These characteristics combined make security code reviews very suited for software that is business critical and relatively small. Being able to gain a high level of confidence that every part of the software is secure can be very valuable when the software is important for the continued existence of the organisation. And in this case the benefits will often outweigh the costs of the review. However, for a lot of other software a security code review will be too expensive. The potential damage from a digital security breach through this piece of software may be much lower than the costs to perform a security code review.

But also for other software it is still important for an organisation to know the vulnerabilities. This is where a combined verification approach of penetration testing and security code review can help. This combination has a high level of synergy, as it takes away many downsides of both verification methods. As a result, it gives a higher number of total findings and less false-positives for a reasonable time/cost when compared to the two verification methods separately. It also allows at least some verification on closed source libraries or binaries used by the software. However, it does require a specific set of skills from the verifier, which may be difficult to find. And for software that is very complex or has a very large code base, it may be better to do a partial security code review or employ other verification methods.

Even though security code reviews have not been used often in practice, we believe that they can certainly add value to the security of software. Especially due to changes in the threat landscape over the past decade, organisations now want to know all their vulnerabilities and security code review can be a great help to reach this goal. And the combination of security code review with penetration testing yields much greater results than the parts for only a small increase in costs and effort, and is therefore here to stay.
Bibliography


[15] ISACA. About ISACA.


[27] Dmarmesh M. Metha.


