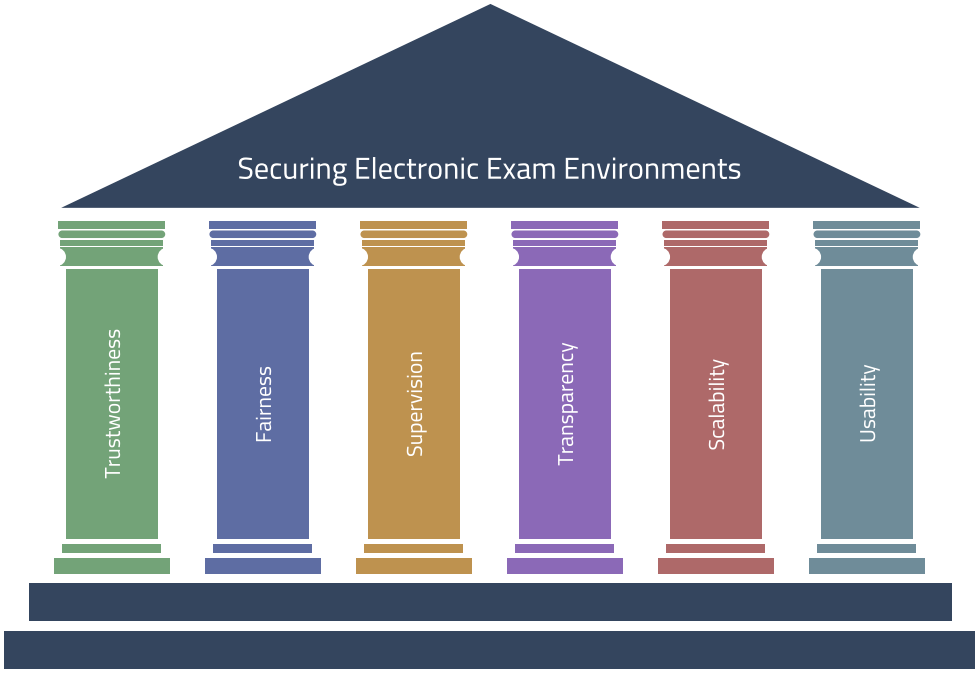

Securing Electronic Exam Environments

Threats, mitigations, and design principles

Master's thesis in Computer Science and Engineering

DANIEL CRONQVIST
SAGA KORTESAARI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Securing Electronic Exam Environments

Threats, mitigations, and design principles

DANIEL CRONQVIST
SAGA KORTESAARI

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Securing Electronic Exam Environments
Threats, mitigations, and design principles
DANIEL CRONQVIST
SAGA KORTESAARI

Supervisor: Mohammad M. Ahmadpanah, Department of Computer Science and Engineering
Examiner: Andreas Abel, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of the six design principle pillars proposed in this thesis.

Typeset in LaTeX
Gothenburg, Sweden 2023

Securing Electronic Exam Environments
Threats, mitigations, and design principles
DANIEL CRONQVIST
SAGA KORTESAARI
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Electronic exams have gained widespread popularity due to their convenience and advantages, particularly in courses involving writing or programming assessments. However, along with their benefits, electronic exams also pose the risk of facilitating cheating, especially when examinees are allowed to use their own devices. To ensure that in-hall bring-your-own-device (BYOD) electronic exams are as secure as their traditional paper-based counterparts, significant measures must be taken to secure the exam environment. This study focuses on two types of e-exam environments: software-based and OS-based.

The thesis presents a comprehensive threat modeling process using the Quantitative Threat Modeling Method (QTMM) to identify various cheating-related threats. Based on these findings, the research proposes eight new design principles to guide developers in creating robust and secure e-exam environments as part of their design strategy. These principles are then evaluated through a case study conducted on a popular e-exam environment, Safe Exam Browser (SEB). The case study reveals several vulnerabilities and successful attacks, highlighting that six out of the eight proposed design principles were not adhered to. To address this problem, the thesis presents a novel design proposal for Safe Exam Browser that aligns with the suggested design principles. Implementation of this proposal would effectively address many of the preventable threats, including a significant design flaw.

Lastly, the thesis explores how well both software-based and OS-based e-exam environments can mitigate threats by following these design principles. By emphasizing the importance of robust security measures in e-exam environments and providing practical recommendations, this research contributes to the ongoing efforts to enhance the integrity of electronic examinations.

Keywords: Security, threat modeling, electronic exams, vulnerabilities, design principles.

Acknowledgements

We would like to thank our supervisor Mohammad M. Ahmadpanah for providing valuable feedback throughout the whole project. It has truly been great working with you.

Thanks to Arne Linde and Emilio Suarez Ardiles, who trusted us with this project and allowed us to investigate the e-exam solution used at Chalmers. Finally, we would like to thank Jonathan Carbol and Hugo Stegrell for providing feedback on the writing and content of this thesis.

Daniel Cronqvist & Saga Kortesaari, Gothenburg, 2023-06-02

Contents

List of Figures

List of Tables

Glossary

BYOD	Bring-Your-Own-Device
CA	Certificate Authority
CAT	Component Attack Tree
CVSS	Common Vulnerability Scoring System
EEE	Electronic Exam Environment
HSM	Hardware Security Module
LMS	Learning Management System
MITM	Man-In-The-Middle
QTMM	Quantitative Threat Modeling Method
RA	Remote Attestation
SEB	Safe Exam Browser
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege
TC	Trusted Computing
TPM	Trusted Platform Module
VM	Virtual Machine

1

Introduction

Electronic exams, also referred to as *e-exams*, are getting more and more popular. They have a wide range of benefits compared to traditional paper exams, such as automating exam correction and allowing examinees to write the exam faster. Unfortunately, these advantages also come with disadvantages; e-exams introduce many new cheating-related threats [1].

These threats are mainly dependant on three factors: *where* the e-exam is taken (what location), which *e-exam solution* is used, and *whether* the exam is proctored or not. As for where, there are a few possibilities: a BYOD (Bring-Your-Own-Device) in-hall e-exam, remote e-exam, or taking the exam in a dedicated computer room. The e-exam solution that is used could either be completely unrestricted and only require that examinees take the exam in a standard LMS (Learning Management System), which is a web page for hosting and writing exams. It could also be more restrictive, and force examinees to use an e-exam environment in addition to the LMS while taking the exam. Thus, when referring to an e-exam solution, it refers to the LMS and an optional e-exam environment. Lastly, whether the exam is proctored or not may differ depending on where it is taken, e.g. in-hall exams are usually monitored by an invigilator and remote exams may be recorded via a camera.

In order to combat these cheating threats, the e-exam solution must implement suitable security measures. In the ideal world, an e-exam should not make it easier for an examinee to cheat in comparison to a paper-based exam. When replacing an in-hall closed book paper-based exam with a BYOD in-hall e-exam, there must be appropriate security measures taken in order to prevent the examinee from cheating. This type of exam typically requires a more restrictive e-exam environment to be more like its paper-based equivalent. The e-exam environment used can be one of two common types: a software-based e-exam environment, such as Safe Exam Browser [2] or an OS-based e-exam environment, such as ExamOS [3]. The most important point here is that both of these types of e-exam environments must be *secure*. Suppose such an environment would contain security vulnerabilities of any kind. In that case, this might allow a student to extend their privileges to something that would not otherwise be possible during a traditional paper exam, allowing them to cheat.

The question is whether these e-exam environments could be made secure as part of their design, to avoid common vulnerabilities. Ideally, the developers creating these

kinds of environments must be sure that they have patched *all* of the potential vulnerabilities, meanwhile, an attacker (an examinee) only has to find one vulnerability in order to cheat in the exam. In reality, 100% secure software is hard to accomplish. Thus, the goal of a developer is to *minimize* the number of vulnerabilities, rather than patch them all.

1.1 Background

This thesis addresses the problem of securing e-exam environments by design. This is usually referred to as *security by design* [4]. Software is often analyzed both during and after implementation, resulting in vulnerabilities being found and patched. It is very common for people to find vulnerabilities in software and then report this to the company building the software, leading to the vulnerabilities being patched. However, if we consider this retrofitting approach in the case of e-exam environments, it might lead to vulnerabilities potentially never being reported in the first place. If a student knows about a vulnerability that allows them to cheat in their exams, they will most likely not report it, which means that the vulnerability will potentially never be fixed. This is why it is very important that we can ensure that these types of environments are secure, ideally in the design phase.

A great example of such an approach shift is the proposal of a set of principles that should be followed when designing cryptographic protocols by Abadi and Needham [5]. The paper has helped cryptographers design more secure protocols, and avoid vulnerabilities that are difficult to spot during the design of protocols. Another example is the paper by Bastys et al. which proposes a set of design principles for information flow control [6]. In a similar fashion to the papers listed here, we propose a set of *design principles* that help secure e-exam environments as part of their design, in order to prevent many common cheating threats. The principles are then used when evaluating the security of the e-exam environment used at Chalmers University of Technology, Safe Exam Browser [2].

Minimizing the number of vulnerabilities in e-exam environments is more complicated than it initially seems since one has to make tradeoffs between *security* and *usability*. It has been previously shown that both security and usability are amongst the most important aspects of an e-exam solution [7]. Focusing too much on the security aspect might make the solution harder to use, which will require e.g. more technical assistance throughout the exam. Yee [8] explains this conflict as very problematic, as developers often see security and usability as each other's complements which causes an immediate conflict during development. It is therefore important that usability and security are part of the design process, and not used as *"magic pixie dust"* on a finished product, as Yee states.

In addition to usability and security, scalability will also be of primary concern, since it has been shown to be an important factor [1]. Scalability refers to how well the e-exam environment scales to large classes in simultaneous examination. For e-exam environments to scale well, it is important for the environment to work in a BYOD setting as the cost of providing workstations for an e-exam in large

classes might grow unreasonably large. Hietanen [3] proposes an OS-based e-exam environment booted from a USB stick, which is much cheaper than providing entire workstations. However, the results show that many examinees had trouble getting it to work because of unfamiliarity with booting an operating system from a USB stick. The solution also introduces several issues scalability-wise, mainly due to three factors: *preparation*, *time*, and *cost*. Each examinee needs to have their own USB stick, which means that someone has to prepare all of these (put the actual OS on it), which in turn requires time and money. The process of updating the OS on each of the USB sticks is also tedious since one most likely has to do it manually. Lastly, the *amount* of USB sticks that are required to conduct such exams could be hundreds, or even thousands.

A software-based e-exam environment will not require any additional hardware or equipment except for the brought device, eliminating those scalability issues. Furthermore, students unfamiliar with booting an OS from a USB stick will not have to do so, instead, they will only be required to install an application before the exam, making the environment more usable. However, the question remains to see whether a software-based e-exam environment can have a similar level of security as an OS environment. This thesis therefore partially addresses if it is possible to achieve a similar level of security between an OS-based e-exam environment and a software-based e-exam environment while addressing usability, security, and scalability as primary concerns.

1.2 Aim

This thesis focuses on proctored in-hall BYOD (Bring-Your-Own-Device) e-exams (electronic exams), and the security risks associated with those types of examinations. Therefore, when we refer to an *e-exam*, this means a BYOD, proctored in-hall exam.

In this thesis, we answer the following research questions:

RQ1: What cheating threats exist for e-exams, and how severe are these?

RQ2: What is a set of common design principles for e-exam environments that, when followed, will ensure that the environment is secure and those cheating threats are mitigated?

RQ3: Will a software-based e-exam environment and an OS-based e-exam environment be able to achieve a similar level of security when following the design principles?

RQ4: Does the e-exam environment at Chalmers University of Technology, Safe Exam Browser, satisfy the design principles and therefore mitigate the found cheating threats?

1.3 Methodology

The methodology employed in this thesis involves a comprehensive literature review to identify research gaps in electronic exam security. Utilizing the Quantitative Threat Modeling Method (QTMM) and the STRIDE framework, we analyze e-exams, by identifying cheating-related threats across its components. A severity scale is established to prioritize threats based on their impact. Design principles are proposed and evaluated through a case study on Safe Exam Browser (SEB), highlighting security flaws and suggesting a new design scheme. This methodology combines literature review, threat modeling, severity assessment, design principle formulation, and case study analysis to comprehensively investigate and secure electronic exam environments, addressing vulnerabilities and enhancing security and integrity.

1.4 Limitations

As mentioned in Section 1.2, the focus of this thesis is specifically BYOD (Bring-Your-Own-Device) in-hall e-exams. In-hall e-exams usually pose less of a threat than remote e-exams, since invigilators are able to continuously monitor the students throughout the exam. It has previously been shown that remote exams introduce new types of threats that can be difficult to deal with. Vegendla and Sindre [9] present some of the threats and conclude that *"most cheating threats become difficult to handle when a written exam is done remotely"*.

An exam can be split into three specific phases: *before, during* and *after*. For electronic exams, these phases have very different characteristics, and thus very different security concerns. This thesis will be focusing on the *during* phase of an exam. This implies that the threats of interest are threats that an examinee can utilize during the exam. This also includes threats that are prepared *before* the exam but used during the actual exam, such as modifying the functionality of the e-exam environment.

1.5 Ethical considerations

Since this thesis contains a security assessment done on Safe Exam Browser, it touches on the topic of *ethical hacking* [10]. Ethical hacking is done with non-malicious intent, where the goal is to find existing vulnerabilities and report them to the stakeholders. Nowadays, many companies welcome ethical hackers to test their systems by participating in so-called bug bounty programs, such as HackerOne[1].

SEB is open source, and freely available on GitHub, meaning that anyone can view the code. With open source, as the name indicates, one is welcome to contribute to the source code. SEB is licensed with the Mozilla Public License 2.0[2], which allows

[1] `https://hackerone.com/`

[2] `https://www.mozilla.org/en-US/MPL/2.0/`

us to perform our penetration test without breaking any licensing rules. We have also made sure to report all of the found vulnerabilities to the developers of SEB, to make them aware of the found issues.

Cheating is unethical and goes against academic integrity. The goal of the thesis is to be able to contribute to the knowledge of building secure e-exam environments, which in turn will make the process of cheating harder. Our hope is that our contribution will make sure that students maintain academic integrity, thus maintaining the corresponding university's reputation. From the early stages of the thesis, we have had close interactions with the e-exam administration at Chalmers to ethically follow the steps of coordinated disclosure. We have reported our findings to them continuously throughout the work to make them aware of any security issues. Any details deemed as sensitive by the e-exam administration have intentionally been excluded from the thesis. Lastly, the information in this thesis was not shared with any other student prior to its publication.

1.6 Thesis outline

The thesis is structured as follows.

Chapter 2 introduces related work within the area of electronic exams and security by design to further motivate the work of this thesis.

Chapter 3 examines the two types of e-exam environments that are of interest in this thesis: OS-based environments and software-based environments. Some differences and similarities between the two types of environments are presented. The chapter serves as an introduction to RQ3, where we later on in the thesis use the contents of this chapter to deem whether the two types of environments can be made equally secure.

Chapter 4 introduces Safe Exam Browser along with its corresponding functionality and architecture. The chapter is meant as an introduction for the reader to understand the technicalities of SEB, which will, later on, be examined further in the case study conducted in Chapter 7.

Chapter 5 examines and outlines what cheating threats exist for an e-exam environment today, related to RQ1. The threats are displayed using attack trees and classified into their corresponding severity levels.

Chapter 6 introduces the design principles related to RQ2. The design principles are a central building block of this thesis and are later on used in the case study conducted in Chapter 7, to evaluate whether Safe Exam Browser can be determined secure.

Chapter 7 conducts a case study on Safe Exam Browser, related to RQ4, which is the e-exam environment used at Chalmers University of Technology. The attacks tested in this chapter are initially presented in the attack trees in Chapter 5. The chapter reveals many security issues present today, along with their possible mitigations. An underlying flaw in the design of SEB is presented, which is further

discussed along with a possible mitigation method in **Chapter 8**.

In **Chapter 9**, the focus shifts towards the examination of OS-based e-exam environments and software-based e-exam environments, addressing RQ3. The chapter delves into the distinct capabilities of these two types of environments in mitigating threats, aiming to uncover their respective advantages and disadvantages. Furthermore, it explores the possibility of achieving equal levels of security in both types of e-exam environments by adhering to the design principles proposed in Chapter 6.

The thesis is concluded and future work is discussed in **Chapter 10**.

2

Related work

This chapter outlines previous work within the area of electronic exams and security by design in order to further motivate our work in this thesis.

2.1 Electronic exams

There have been a fair amount of papers examining different aspects of electronic exams, ranging from examining the security of certain solutions, to identifying how efficient they really are compared to paper-based exams.

The doctoral thesis by Chirumamilla [7] studies security threats and requirements in e-exams. A comparative analysis between paper exams and e-exams is presented [1], which has been useful for us when identifying threats related to e-exams, relevant to RQ1 of the thesis.

Recently, in 2021, Hietanen wrote his Master's Thesis *Security of electronic exams on students' devices* [3]. The thesis investigates how to secure BYOD e-exam environments. Hietanen identifies potential threats in different e-exam environments along with possible mitigations. To counter the identified threats, he develops an OS-based e-exam environment, ExamOS, which consists of a hardened operating system with controlling software running on it. The end result shows that ExamOS is successfully able to mitigate most threats. However, the result showed that ExamOS was technically difficult to use for some examinees resulting in a "significant amount" of technical assistance required during the exams. This assistance, unfortunately, delayed the starting times of examinations as well. Our view is that the environment should be seamlessly easy to use for an examinee, and not cause delays due to examinees' unfamiliarity with technicalities. Often, by introducing too many security measures, a system becomes difficult to operate [8]. Chirumamilla [7] compared multiple case studies on the topic of usage of e-exam solutions and showed that the most important aspects of an e-exam solution were *usability*, *marking*, *security* and *reliability*. Thus, usability is an important aspect, and it seems like Hietanen underestimated this fact in his thesis, or overestimated the examinees' ability to boot an operating system from a USB stick without assistance.

Our belief is that a BYOD e-exam environment becomes easier to use for an examinee if it is software-based, similar to Safe Exam Browser [2]. There are multiple advantages with this solution, such as it being *easier* to download a standalone

program than booting a new OS from a USB stick, and it being more cost-efficient. *Easier* here means that it is likely easier for the average student to download a program and run it on their computer than to boot an operating system from a USB stick. The current downside of this environment is that it brings on new types of security-related threats since it runs on the examinee's own laptop, with their own operating system. The point here is that both types of environments contain pros and cons. This point is further examined by us in Chapter 9, where we investigate whether both of the two environments can be made equally secure by design, which is related to RQ3 of the thesis.

2.2 Security assessment of older Safe Exam Browser versions

The security of Safe Exam Browser [2] has been examined prior to this thesis through different perspectives. The doctoral thesis by Chirumamilla [7] is an example of such, where they perform a penetration test using the HARM method [11, 12] on Safe Exam Browser. Heintz [13] and Søgaard [14] also identify the security flaws of the application. The results of the three papers show that Safe Exam Browser does indeed contain security flaws.

An important detail is that after all of these three penetration tests were conducted, on May 28th 2020, version 3.0.0 of SEB was released where it was *completely rewritten from scratch* [15]. The update included several new features along with a completely new embedded browser engine. We have not found similar case studies or testing on this version of SEB, and our work aims to fill that gap by performing a thorough security analysis on this new version of SEB, related to RQ4 of the thesis. The papers mentioned have served as a baseline for our case study on Safe Exam Browser presented in Chapter 7, where a few of the older established attacks have been attempted to see whether the developers have patched the flaws in newer versions.

Since software is continuously developed, it is important to continuously assess its security, since newer updates may introduce new types of security flaws. It might even be that newer versions open up the possibility of exploiting old flaws that were previously patched, due to change of design in the application.

2.3 Security by design

Security by design is the concept of taking security into account all the way from the design process, with the goal of designing foundationally secure applications. The concept is important due to security often being overlooked in the development process, resulting in it being de-prioritized which later on leads to vulnerabilities being found and exploited in the end. Security is first and foremost a design consideration, meaning that it isn't something that you graft on at the end [4].

The authors of *Secure by design* [4] present some very good points in regard to

why security is an important part of the design process. To start with, requiring developers to constantly think about security while working is not realistic since far from everyone is proficient in the area. Requiring everyone to write secure code would assume that every developer is a security expert, as well as assuming that they can think of every potential vulnerability that might occur now or in the future. Rather than always actively focusing on the security flaws when developing the software, a better approach is to shift focus onto the *software design*. This means that security should be incorporated into the design process, which actually introduces the benefit of non-security experts *naturally* writing secure code, due to the design of the system implicitly avoiding insecure constructs.

Design principles When designing secure applications, it is common for developers to adhere to and follow design principles that help them make more secure decisions. The paper *Prudent Engineering Practice for Cryptographic Protocols* by Abadi and Needham [5] is an example of such, where they present design principles that help secure cryptographic protocols. The paper made a breakthrough at the time of writing, where it helped discover many major security flaws present in cryptographic protocols at the time. For years, it has served as a guideline for developers designing these types of protocols, ultimately eliminating many common decisions that may lead to security flaws.

According to our knowledge, no such design principles exist for e-exam environments prior to this work. Therefore, similar to the papers described above, our work presents design principles related to making e-exam environments *secure by design*.

3

E-exam environments

It is very common for an e-exam solution to require an e-exam environment in order to *secure* an exam. The purpose of the e-exam environment is to prevent and/or detect an examinee from accessing prohibited material during an exam. In the case of BYOD in-hall exams, we believe that an e-exam environment is a must since otherwise, an examinee would have full access to their device which implies access to cheating materials.

In this chapter, two different types of common e-exam environments are examined: software-based environments, and OS-based environments. There are multiple differences between these two environments, such as how they are implemented and what types of threats they can counter. Another key difference between environments is whether they are open-source or not. Publishing an environment in an open-source manner opens up the possibility for other people to easier examine the environment and report any security vulnerabilities they find. Like *Linus's Law* [16] states: "given enough eyeballs, all bugs are shallow". However, the disadvantage is that malicious actors now have an easier time spotting vulnerabilities and utilizing them to cheat. This is a common challenge with open-source software, but it is generally believed that the benefits outweigh the disadvantages.

The goal of the chapter is for the reader to successfully be able to identify the difference between a software-based environment versus an OS-based environment. The two types of environments are further examined in Chapter 9 of the thesis to identify whether both of them could be made equally secure, or if one of them contains flaws that cannot be solved in a BYOD e-exam setting, related to **RQ3**.

3.1 Paper-based exams versus e-exams

While e-exams offer significant advantages over paper-based exams in certain aspects, they are limited in their ability to mitigate certain types of threats. Threats that are executed through physical means, such as bringing a paper cheat sheet [17], are impossible to prevent. Therefore, in-hall invigilators will still be required for e-exams, to help mitigate such threats.

It is crucial to acknowledge these limitations of e-exam environments and incorporate appropriate security measures, both as part of the environment itself and also in the physical world. While e-exams provide a more flexible and efficient alterna-

tive to paper-based exams, they are certainly not foolproof, and new threats and vulnerabilities may emerge over time. As a consequence of this, it is essential to consider these limitations when designing and implementing e-exam environments and to continually evaluate and improve their security measures to enhance their ability to mitigate cheating threats.

3.2 Software-based environments

Software-based environments are installed as an application on an examinee's device. Some of these environments utilize so-called *kiosk mode* functionality, which locks down an examinee's computer and prevents them from accessing prohibited materials during an exam. However, there exist other software-based environments that have taken a completely different approach. Instead of locking down the examinee via kiosk-mode, these environments utilize extensive logging, meaning that an examinee would have full access to their device but the application instead logs everything they are doing. These two types of approaches can be regarded as *active* versus *passive* in terms of preventing cheating. The kiosk mode approach *actively* prevents an examinee from cheating, whereas the other approach does not prevent an examinee from cheating making it more *passive*. However, even if the second approach can be regarded as more passive, an examinee can still be caught cheating via auditing of the logs.

In this section, we present a couple of different software-based e-exam environments, with the purpose of showing the reader what types of environments exist today. We also outline the differences between them and discuss the different challenges that need to be solved depending on *how* the environment is designed.

3.2.1 Safe Exam Browser

Safe Exam Browser [2], SEB, is an open-source lock-down-based browser that is commonly used today. This section will present a very brief overview of SEB. SEB is further examined in Chapter 4 and Chapter 7, where we perform a case study on the environment in order to find current vulnerabilities in it. The application utilizes kiosk mode functionality that prevents a user from accessing prohibited materials during an exam. This means that SEB utilizes an *active* approach to prevent an examinee from cheating. SEB is configured via a configuration file, used to apply certain settings to the environment, such as what third-party software is allowed to run and what URL the browser should show the examinee during the exam. The software is examined in greater detail in Chapter 4, where the architectural overview can be found in Figure 4.1.

In order to prove to an LMS that an examinee is using SEB during an exam, SEB uses three HTTP headers which are appended to all network requests. The headers are calculated using a combination of the configuration file and the executable, which is further discussed in Section 4.5. The LMS must then verify these headers to know whether an examinee is using an unmodified, legitimate version of SEB.

SEB logs activity to a pre-defined log file that contains information about what web pages the examinee has opened and what OS version the machine is running. The logs are not transmitted to any server, except if the exam requires an examinee to connect SEB to *SEB Server* [18]. The logging functionality of SEB as well as SEB Server are further described in Section 4.6 and Section 4.7.

3.2.2 Digiexam

Similar to Safe Exam Browser in Subsection 3.2.1, Digiexam [19] is a software-based e-exam environment that can conduct exams in a lock-down mode. The major difference is that Digiexam is closed-source, which also implies that it has been hard for us to find material related to *how* Digiexam works.

Digiexam is being marketed as *Secure* and *Compliant* [20]. *Secure* refers to the application having "advanced and secure device lockdown", while *Compliant* refers to the application being "fully compliant with GDPR". However, since the application itself is closed-source, it is hard to actually confirm that what they are claiming is true. In the case of digital exams where an examinee is required to download software to their computer, it is often beneficial for the software to be open-source so that the developers are transparent with what is running on the examinee's machine.

3.2.3 The FLEX framework

Küppers et. al. [21, 22] propose the FLEX framework intended for conducting secure e-exams. The framework includes an application that will be installed on an examinee's system, in other words, a software-based e-exam environment. However, instead of utilizing lock-down techniques, the environment instead relies on logging for retrospective auditing of cheating behavior. This means that a student may exit the environment, but since the environment is monitored, the action will be logged to a server. This sort of approach shift is very interesting, as other solutions rely heavily on the lock-down mechanism instead.

The reason why FLEX relies on logging rather than lock-down is described further in a paper by the authors [21]. According to them, it seems "nearly impossible" to implement a lock-down-based software that locks down all operating systems in the same way. The authors further argue that the logging mechanism is similar to the approach used throughout paper-based exams: you cannot prevent a student from bringing a cheat sheet into the exam hall, but an invigilator may notice this and remove the cheat sheet from them. The same goes for the approach they use for the e-exam environment, an examinee *can* access prohibited materials such as cheat sheets on their device, but the logging will actively spot them doing so.

In order to prevent malicious modification of the application, its integrity has to be verified. The integrity verification of the FLEX application utilizes *Remote Attestation (RA)*. Remote Attestation is a part of trusted computing, which allows a verifier to verify the state of a remote client [23]. Remote attestation can be done in many ways, either purely through software or with the help of hardware security modules present in modern devices today. Not only does RA verify the integrity of the ap-

plication, but it also verifies that an examinee is indeed running the application in the first place, meaning that it is unfeasible to circumvent it completely.

3.3 OS-based environments

Even though operating systems are classifiable as software as well, an OS-based environment is one that occupies the entirety of the operating system. This means that the operating system's primary intention is to only facilitate enough functionality to be able to do the exam. It is naive to assume that all examinees are able to install a second operating system on their machine by themselves, which is why environments like these are commonly distributed through bootable USB sticks.

Similar to software-based environments, OS-based environments can also be viewed as lock-down environments. An OS-based environment is not locking down the device itself, but it is effectively *isolating* an examinee from accessing any other material on their device, by forcing them to take the exam in that specific OS.

3.3.1 ExamOS

Hietanen presents the hardened operating system *ExamOS* (Exam Operating System) with controlling software running on it to secure the system against cheating examinees [3]. The source code for ExamOS and all of its related parts can be found on GitHub [24], meaning that it is open-source. ExamOS was developed as part of Hietanen's Master's Thesis as an attempt to secure e-exams at Aalto University.

ExamOS is a Linux-based operating system with a set of carefully selected hardware peripheral drivers and other OS-level functionalities. Hietanen argues that the level of control one has over a custom operating system allows him to be confident that no examinee will be able to cheat successfully.

As part of ExamOS, Hietanen also developed a software called *Exam-tool*. Exam-tool is described as "a multi-application package that consists of background services and software that the examinee interacts with during an exam". As part of exam-tool, a standalone *Exam Browser* was developed. The Exam Browser is a restrictive browser that only allows an examinee to access specific sites allowed in the exam configuration. The exam configuration is provided to the Exam Browser by the *Exam Service*, which configures the ExamOS system according to an exam's configuration. The Exam Service fetches this configuration from a specific *Exam server*. Along with fetching the configuration, the Exam Service is also responsible for communicating all logged data to the Exam server. Such data is for example the hardware identification data to prevent multiple different examinees from authenticating to the system with the same credentials. An overview of the Exam-tool package is shown in Figure 3.1.

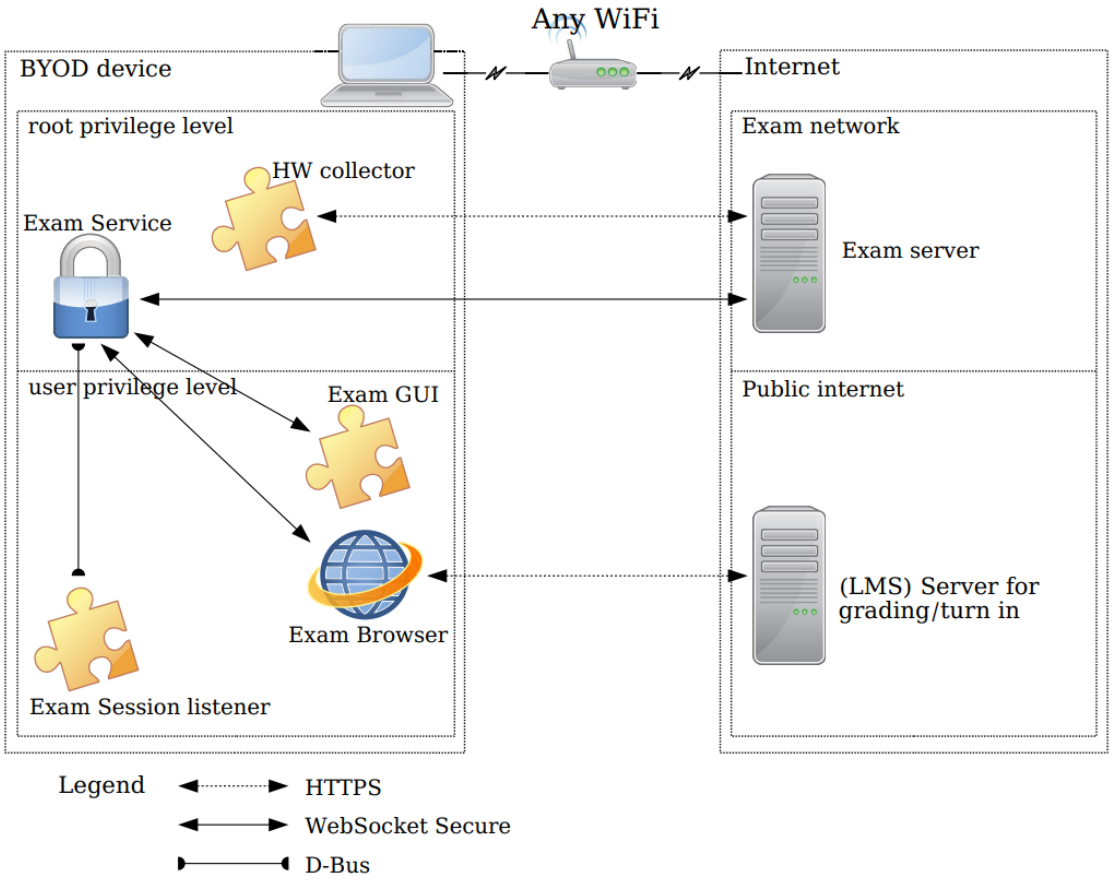

Figure 3.1: Overview of the Exam-tool package [3].

ExamOS can essentially be seen as an OS-based e-exam environment with a software-based e-exam environment running inside of it. When conducting exams with ExamOS, the examinees are given USB sticks that need to be plugged into their devices in order to boot up the operating system.

As previously mentioned in Section 2.1, ExamOS was technically difficult to use for some examinees, who required technical assistance from invigilators, which resulted in significant delays throughout the exam process. This was most likely due to the examinees' unfamiliarity with ExamOS.

3.3.2 Australian national e-exam project

As part of the "national e-exam project" in Australia, Edith Cowan University trialed an open source OS-based e-exam environment at their university throughout 2016 [25]. The e-exam environment is described as an "enclosed computer-based environment that is isolated from the internet and any resources other than those provided by the lecturer". The OS is a modified version of Ubuntu that prevents internet, Bluetooth, and local drive access, along with a custom "exam starter" that guides students to begin the exam [26].

In contrast to ExamOS described in Subsection 3.3.1, this OS-based environment

does not store examinees' answers on a server. Instead, all examinees need to hand in their USB sticks after an exam, which are later used by the examiner in order to retrieve the exam responses. Figure 3.2 shows the flow of the environment. As seen in the figure, a USB duplicator is used in order to upload exam contents onto the USB sticks, as well as when retrieving the exam responses from them. Even though the USB duplicator might have made the process of copying the material onto the USB sticks slightly faster, it is mentioned that "the USBs were then manually checked to ensure all files had been copied correctly" [25].

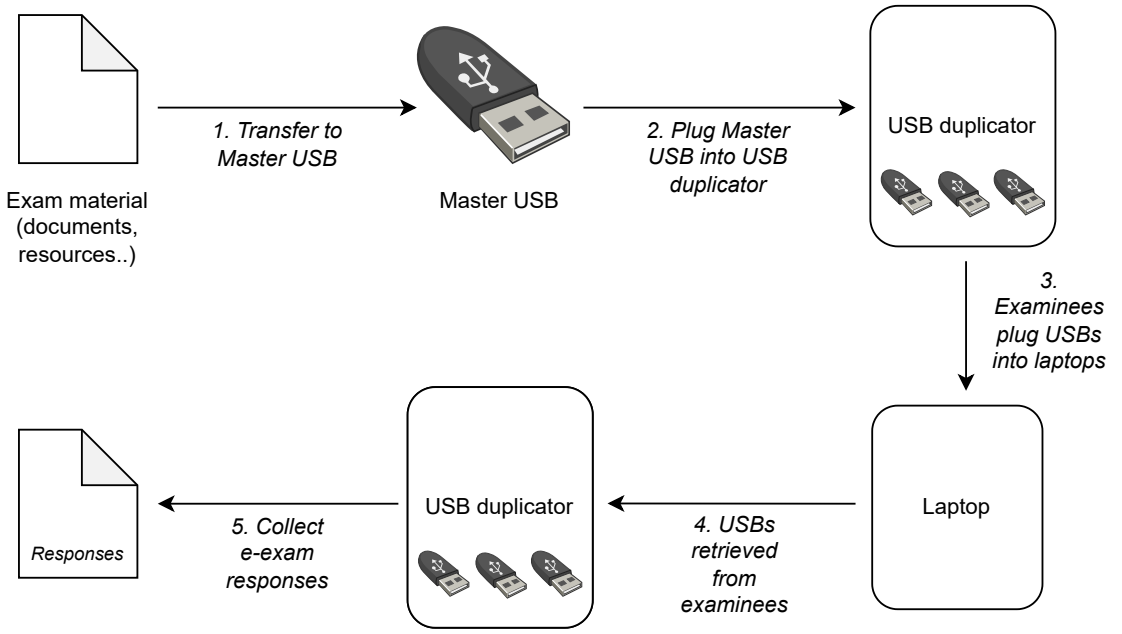

Figure 3.2: Overview of the OS-based e-exam environment used at Edith Cowan University in 2016.

The environment does not utilize passive detection of any kind, such as logging. This is most likely due to the operating system being completely isolated from the internet, as described above. However, since the USB sticks are retrieved from the examinees, a possibility would have been to implement local logging that could be stored on the USB, which could be checked when the USB is retrieved from the examinee.

It was reported that students were hesitant to use the e-exam environment due to distrust around booting up their machine with a different operating system along with a fear of losing data. The end result also showed that a few students had trouble navigating the OS, due to unfamiliarity with it [25].

3.4 Logging

As seen from the different environments described above, it is very common for e-exam environments to utilize logging of some kind. Logging can be seen as a *passive* approach towards detecting cheating: it might not *actively* prevent an examinee

from cheating, but it can notice if they are doing so. Most e-exam environments utilize a combination of lock-down and logging, but as seen from the examples above there are some environments that use only one of them.

Logging of personal data on an examinee's device may potentially infringe on their privacy. It may also be that some type of logging may be prohibited in certain areas due to privacy laws and regulations, such as General Data Protection Regulation (GDPR) [27], which is EU's data privacy and security protection law. This may potentially introduce additional challenges to developers of e-exam environments since they have to keep these regulations in mind. In the case of GDPR, the developers must make sure to follow Article 17 (*Right to be forgotten*) [28], which states that people should have the right to request personal data to be deleted. It may also be that some types of logging may require written consent from examinees, according to laws and regulations.

3.5 Environment comparison

Here we summarize the environments described in the chapter, and present their discussed features, advantages and disadvantages in Table 3.1. As introduced in Section 3.2, the terms *passive* and *active* represent the environment's method of mitigating examinee's from cheating. Active environments typically implement some kind of lock-down solution to prevent cheating, whereas passive environments often rely on logging to retrospectively check if cheating was done. As the table shows, and the previous sections describe: some environments are active, some are passive, and some implement solutions that can be categorized as both.

Name	Type	Passive	Active	Open-source
Safe Exam Browser [2]	Software	✓	✓	✓
Digiexam [19]	Software	✓	✓	✗
FLEX Framework [21, 22]	Software	✓	✗	✗
ExamOS [3]	OS	✓	✓	✓
Australian national e-exam project [25]	OS	✗	✓	✓

Table 3.1: Summary of the different e-exam environments presented in Chapter 3.

4

Safe Exam Browser (SEB)

Safe Exam Browser, also known as SEB, is an e-exam environment used at Chalmers University of Technology and across many other universities around the world. This chapter will dive into the technical details of SEB, such as what it does and how it works.

4.1 Architecture

Citing from the SEB website [29]:

> *"Safe Exam Browser is a web browser environment to carry out e-assessments safely. The software turns any computer temporarily into a secure workstation. It controls access to resources like system functions, other websites and applications and prevents unauthorized resources being used during an exam."*

SEB is an open-source [30] kiosk mode application. As seen in the architecture diagram in Figure 4.1, the SEB kiosk application contains an integrated browser that displays a web page from a given URL. Additionally, one or several optional third-party application(s) can be started and run during the exam. It is up to the exam administrator to configure the URL and permit third-party applications, by defining this in the *configuration file*. This file is used within SEB to specify certain settings, which is further explained in Section 4.2.

SEB is available across multiple platforms, such as Windows, MacOS and iOS. In this thesis, only the MacOS and Windows versions will be considered. The Windows version is written in C# and uses a Chromium-based browser engine[1], and the MacOS version is written in Objective-C with a WebKit-based browser engine[2].

4.2 SEB configuration

The *configuration file*, also referred to as *config file*, is used to specify how SEB should function during an exam.

[1] `https://chromium.org/`
[2] `https://webkit.org/`

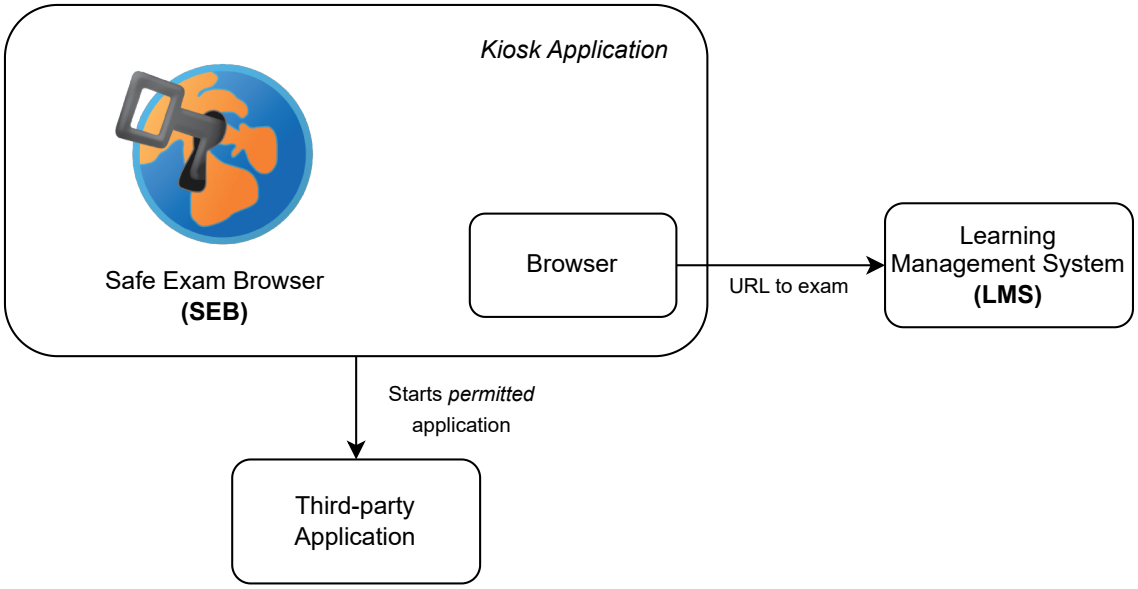

Figure 4.1: Safe Exam Browser (SEB) Architecture overview.

The Windows version of SEB contains a *configuration tool*, `SEBConfigTool.exe`, which is used to create configuration files or configure a local client [31]. For MacOS you can access the tool by opening SEB and clicking *Preferences* in the menu [32]. A screenshot of the `SebConfigTool.exe` can be seen in Figure 4.2, whereas the MacOS-specific tool can be seen in Figure 4.3.

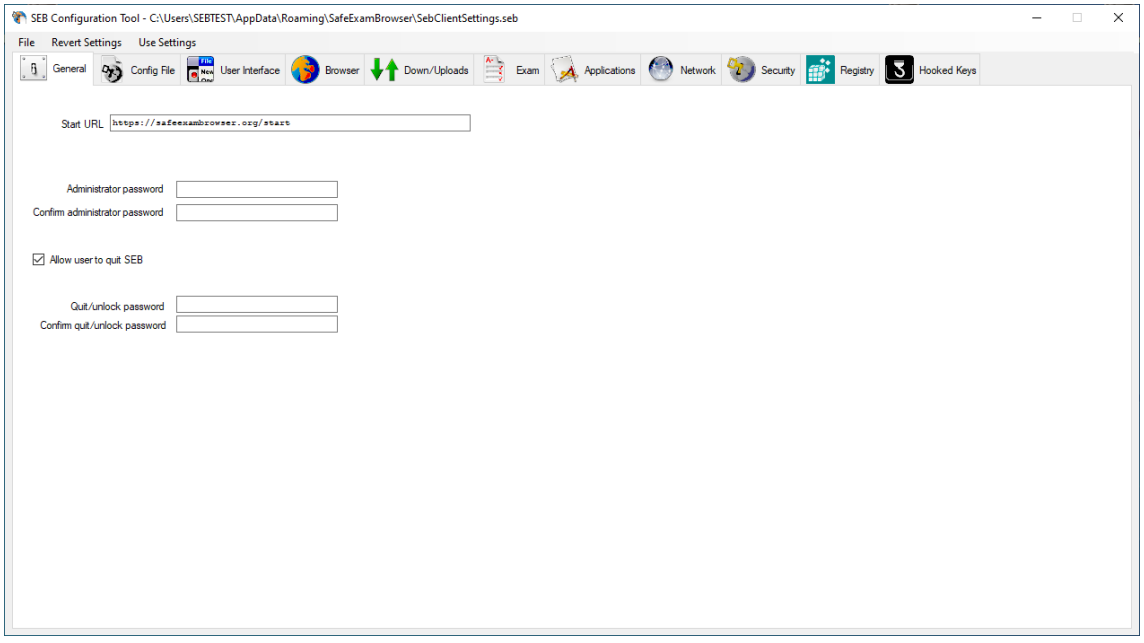

Figure 4.2: Screenshot of the SebConfigTool for Windows.

Figure 4.3: Screenshot of SebConfigTool for MacOS.

Both versions contain the same configuration options for SEB. However, a few configuration options are platform-specific, which means that MacOS-specific settings do not apply to a Windows computer, and vice versa. Even though some options are platform specific, the same config file can be used on all platforms. A few of the configurable options of SEB are shown in Table 4.1. The full list of configurable options will not be included, but the curious reader can find the full list in the SEB documentation for Windows and MacOS [31, 32].

The configuration file that is used can be encrypted using a password, by specifying the settings password shown in Table 4.1, to ensure that none of the configuration options can be easily extracted prior to the exam. If the config file has been encrypted, the examinee needs to enter the password that decrypts the file upon starting an exam that is using the specific config file. SEB uses the open-source RNCryptor framework[3] for encrypting config files [33]. RNCryptor is an open-source implementation of AES [34] with support for many different types of usage, where symmetric encryption with a password-based key is one of them. Using PBKDF2 [35], an encryption key is generated from the given password and is then used to encrypt the config file. Decryption only requires that the password is known since PBKDF2 will generate the same key if the same password is entered. Therefore, if someone knows the password that was used when encrypting a config file, it is trivial to retrieve the decrypted version. The SEB documentation includes a guide on the decryption procedure [36]. The decrypted version of a SEB config file is simply an XML `.seb`

[3] `https://github.com/RNCryptor/RNCryptor`

Key	Description
Start URL	The exam URL that SEB will show the examinee throughout the exam
Quit/unlock password	The password that the examinee has to enter when trying to quit SEB
Settings password	Encryption/decryption password for the .seb config file, also known as *encryption key*
Use Browser Exam Key and Config Key	Yes/no checkbox. If checked, SEB will use these keys to generate the correct headers sent in the network traffic of SEB, see Section 4.5
Allow SEB to run inside virtual machine	Yes/no checkbox. If checked, SEB will be able to be run in a VM
Permitted Processes	Allowing processes to be run throughout the exam
Prohibited Processes	Prohibit processes to be run throughout the exam

Table 4.1: Safe Exam Browser configuration options.

file that specifies the configured settings as key-value pairs, following the Apple plist (property list) format. A small snippet of such a file is shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
    "https://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>originatorVersion</key>
    <string>SEB_Win_2.1.1</string>
    <key>startURL</key>
    <string>https://examurl.com</string>
    - Rest of the key-value pairs -
  </dict>
</plist>
```

4.2.1 Third-party applications

As previously mentioned, third-party applications can be configured to be permitted to run in the config file. By allowing such an application to be run, it can be used normally within SEB, while maintaining the lock-down mode. The examinee may switch to one of the permitted applications while in SEB simply by clicking its icon in the lower bottom taskbar.

These third-party applications may pose a risk to SEB depending on what kind of applications are permitted to run. For example, Microsoft Excel might be a

permitted third-party application during an exam. This would make it possible for an examinee to make use of the many functions that allow a user to retrieve information from outside the Excel environment.

To permit a third-party application during an exam, the config file that is created must simply contain the name of the executable file that is allowed in its list of permitted processes, e.g. `excel.exe`. Additional configuration can be done for each third-party application such as appending certain arguments to the executable, or even allowing the examinee to select where the executable is located before starting the exam. However, according to the documentation of SEB, and after testing, it seems only Windows has support for permitted third-party applications, meaning that not all platforms are able to benefit from this feature [37].

Additionally, it is also possible to prohibit applications from being run at the same time as SEB. If a prohibited application is running when SEB starts, it will automatically be closed upon startup. If an application that is prohibited starts in the background when SEB is running, it will instantly be killed [31]. SEB facilitates a default list of prohibited third-party applications, which typically are applications that may help an examinee to cheat.

4.3 The SEB URL schemes

SEB facilitates a custom URL scheme `seb://`, which makes the process of loading an exam configuration more accessible for examinees [38]. This URL scheme allows an LMS to provide hyperlinks such as `seb://example-lms.com/config.seb` which when clicked will open SEB with the configuration in the file that exists at `http://example-lms.com/config.seb`. The configuration is downloaded and temporarily loaded to be used for this SEB process but is not stored on the host. This makes it easy for examinees to start their exam in SEB with the correct configuration.

4.4 Learning Management System (LMS)

The *Learning Management System*, also referred to as the *LMS*, is where an examinee takes the exam. As previously mentioned, the URL of the LMS is equal to the `startUrl` defined in the config file, as seen in Table 4.1.

Popular LMS choices are Moodle[4], Inspera[5] and Exam.net[6], among others. All of these three learning management systems may differ slightly in what they offer in terms of the exam, that is, one of them might be superior for specific types of exams such as programming exams. The important factor for *all* of these systems is that they all offer the option of taking an exam in a lock-down mode using SEB. Thus, SEB is a popular choice among widely used learning management systems.

[4] `https://moodle.org/`
[5] `https://inspera.com/`
[6] `https://exam.net/`

The LMS is responsible for providing a config file, see Section 4.2, so that the corresponding exam can be started securely in SEB. Typically, an examinee will click a hyperlink that uses the SEB URL scheme mentioned in Section 4.3, which then downloads the config file and opens up the exam in SEB.

Furthermore, the LMS is a very important factor in *how secure* an e-exam will be, when taken in SEB. That is, an LMS needs to make sure to use all of the tools that SEB provides in order to verify that an exam is really taken in SEB. The SEB documentation includes a comprehensive guide [38] of how to verify that an examinee is using SEB. More specifically, the guide states the following verification steps:

1. Make sure an exam can only be taken using Safe Exam Browser. Display an error message if trying to open the quiz/exam in another browser.
2. Check if legitimate SEB settings and the correct version of SEB are used.
3. Quit SEB (and/or unlock the device) automatically after the exam was submitted.
4. Facilitate starting SEB with the correct settings for the exam.
5. Don't display any links inside an exam which would allow to navigate to other sections of the LMS or even other websites.

Making sure that an exam can only be taken using SEB and checking if legitimate SEB settings/version is used can be done by verifying the network traffic that SEB is sending. There are three headers available that authenticate that an examinee is using SEB: `User-Agent`, `x-safeexambrowser-configkeyhash`, and `x-safeexambrowser-requesthash`. Details of how the headers are constructed in SEB can be found in Section 4.5.

To make sure that an examinee is only able to quit SEB after an exam has been submitted, a *quit/unlock password* (see Table 4.1) needs to be defined in the config file. This password would then need to be communicated to an examinee once a trusted invigilator has confirmed that the examinee has submitted their exam.

In order for the LMS to be able to facilitate starting SEB with the correct settings for the exam, it needs to be able to create config files (see Section 4.2), which are automatically started in SEB via the SEB URL scheme (see Section 4.3).

Finally, the exams created through the LMS should not contain any links that would allow an examinee to exit from the exam environment. If the exam would contain any links that would allow an examinee to do so, this could lead to the examinee being able to access material that would otherwise be prohibited during the exam.

4.5 Network traffic

The network traffic that SEB generates has three particularly important HTTP request headers, in order for an LMS to authenticate that SEB is used. Example values of the three headers can be seen in Table 4.2.

HTTP header	Example value
`User-Agent`	`Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.68 SEB/3.4.1 (x64)`
`x-safeexambrowser-configkeyhash`	`dce0ce79d87e7c98b4fa23a66bea227ad15100f4fad7d5e9da2be2eb35157a67`
`x-safeexambrowser-requesthash`	`e50720a375958db595a05525b0d3dab921b1859ca0788a0caf459b97811d45b4`

Table 4.2: HTTP request headers present in requests generated by SEB.

For the `User-Agent` header, SEB appends the `SEB/version` text. This header will always have the same value, meaning that it will not change. The only part of the header that may change is the version number, which depends on what version of SEB is currently used. The `x-safeexambrowser-configkeyhash` and `x-safeexambrowser-requesthash` headers are SHA256 hashes of the request URL appended with either the `Browser Exam Key` (for the `requesthash`) or the `Config Key` (for the `configkeyhash`). This implies that the values of these two hashes will change depending on what URL has been requested.

The `Browser Exam Key` and `Config Key` are generated within SEB if the option *Use Browser Exam Key and Config Key* has been checked in the config file, see Table 4.1. These two keys are slightly different and can be used to verify slightly different aspects of SEB. The code snippet of calculating the `Browser Exam Key`, also known as `BEK`, is shown in Figure 4.4. The code can also be found in the `KeyGenerator.cs` class on GitHub[7]. As seen in the code snippet, the process of calculating the `BEK` is simply taking a SHA256 hash of the SHA1 fingerprint of the code signature for the executable, the file version, and finally the `Config Key`. This implies that the `BEK` will vary depending on what version of SEB is used. It will also differ depending on the platform that the examinee is using, meaning that the `BEK` on Windows will be different from the one on MacOS. This implies that the `x-safeexambrowser-requesthash` header is able to verify two things: the correct version of SEB and the correct configuration of SEB.

[7] `https://github.com/SafeExamBrowser/seb-win-refactoring/blob/b69280731a212cad699f911f98431c5d47104d7b/SafeExamBrowser.Configuration/Cryptography/KeyGenerator.cs`

```
using (var algorithm = new HMACSHA256(salt))
{
    var hash = algorithm.ComputeHash(Encoding.UTF8.GetBytes(
            appConfig.CodeSignatureHash +
            appConfig.ProgramBuildVersion +
            configurationKey));
    var key = ToString(hash);

    browserExamKey = key;
}
```

Figure 4.4: Code from SEB that computes the Browser Exam Key.

The `Config Key`, on the other hand, only depends on the currently loaded configuration of SEB. It is calculated by first converting the config file into `SEB-JSON`, which is a `JSON`-like representation of it [39]. All keys in the `SEB-JSON` representation need to appear in sorted order, some keys are left out for version independence, and no whitespace should occur in the representation. After the `SEB-JSON` has successfully been generated, the resulting `Config Key` is simply a SHA256 hash of it. Thus, the `Config Key` has version and platform independence, meaning that all versions and platforms of SEB will generate the same key. LMS solutions can also calculate the same hash, by performing the exact same procedure.

4.6 Logging

According to the privacy statement [40] found in the SEB documentation, SEB does not collect any unnecessary personal information on an examinee's computer. Some personal data, such as device type/name, OS version, computer username, and URLs of opened web pages can be found in the log files that SEB saves on the system. By default, these log files are not transmitted to any other server, they are simply used for debugging purposes. If anything goes wrong during the startup of SEB, error messages can be found in these files which can help a user debug the underlying issue.

An SEB client can be configured to connect to an instance of SEB Server [18], where the previously mentioned log files will be sent. However, this option is considered as an add-on for SEB, and needs to be specifically configured. SEB will not transmit the log files by default to any other server. The server needs to be set up by the institution in order for the examinees to be able to connect to it. The functionality of SEB Server is further described in Section 4.7.

4.7 Safe Exam Browser Server (SEB Server)

The *Safe Exam Browser Server*, *SEB Server*, is a web application with the purpose of simplifying and centralizing the configuration of SEB clients for exams. The application interacts with an LMS in order to set up and configure exams. According to the documentation [18], SEB server "improves security", by allowing SEB clients to be monitored in real-time during an exam. An architectural overview of SEB Server can be seen in Figure 4.5, where it is shown that the server communicates with the SEB client(s) and the LMS.

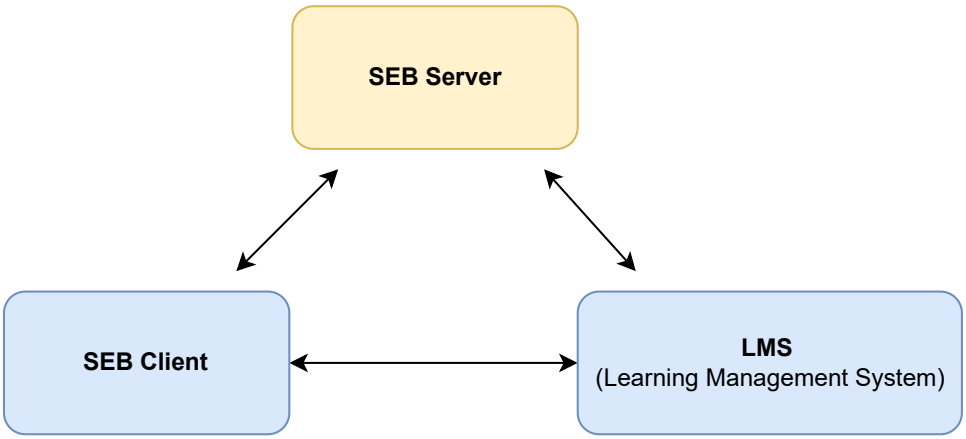

Figure 4.5: SEB Server Architecture overview.

As mentioned, SEB clients can be monitored via the SEB Server during exams. The information available about a client is the same information collected in the log files described in Section 4.6. SEB Server also pings the clients continuously and assigns each client a state. The different states can be one of *Connection Requested, Active, Missing, Closed, or Canceled*. The details of each state are further described in Table 4.3.

State	Description
Connection Requested	Appears whenever a SEB client has contacted the SEB server, but not yet finished the initial connection handshake or logged into the LMS.
Active	Appears after a successful handshake and login into the LMS. Stays as long as the connection is connected, implying that it is not closed or terminated.
Missing	Appears when a SEB client is active, but has missing ping(s).
Closed	Appears whenever a SEB client closes connection, after being active.
Canceled	Appears whenever a connection has been canceled.

Table 4.3: Possible states assigned to each SEB client via SEB server [18].

In order to configure a SEB client to connect to SEB Server, a specific connection configuration file has to be provided to the client. Similar to the configuration file mentioned in Section 4.2, the connection configuration file can be encrypted with a password, which the examinee needs to enter upon startup of SEB.

5

Threat modeling

Threat modeling [41] is the process of identifying the underlying threats of a system in order to gain a valuable understanding of how to mitigate these threats. Once the threats have been identified, one can then establish *security requirements* that need to be followed in order to mitigate the threats in question. The process of threat modeling is important to take into consideration while designing systems and applications. Not only does it help you to find security issues early, but it also helps you to understand your system in-depth and its security requirements [42].

This chapter aims to identify cheating-related threats in e-exams. An important detail to keep in mind is that this thesis focuses on threats that an attacker may use *during* an actively invigilated exam. This means that some types of threats are out of scope, such as those only relevant after the exam has finished (i.e. altering your answers once the exam has finished). However, threats that are prepared *before* the exam takes place are still relevant, since they pose a threat throughout the exam (i.e. modifying an e-exam environment's code before the exam begins, in order to gain an advantage). In order to identify such threats, we utilize the "Quantitative Threat Modeling Method", which is described in detail in Section 5.1. The identified threats are later used when proposing our design principles in Chapter 6, as well as when assessing the security of Safe Exam Browser in Chapter 7.

5.1 Quantitative threat modeling method

The Quantitative Threat Modeling Method, also known as "quantitative TMM" or "QTMM", is a threat modeling method proposed by Potteiger et al [43]. The method involves defining *components* within the system or procedure, thereafter, threats are discovered for the corresponding components. Two approaches are used in order to discover the threats: STRIDE and Component Attack Trees (CATs), which will be further described below. At last, the threats are classified into different severity levels using the Common Vulnerability Scoring System [44], usually referred to as CVSS. An alternative could be to use DREAD [45] for the severity scoring, which is a similar quantitative scoring system. However, as suggested by QTMM, we will use a scoring system similar to CVSS, although with slight modifications due to CVSS containing categories that are non-applicable to the setting. This scoring system will be further described in Section 5.2.

5.1.1 Components

In order to define the components of a system or procedure, one must divide the system or procedure into different parts. This involves identifying which discrete parts a system has, and how they interact with each other. As different systems or procedures vary in complexity and scale, their corresponding destruction into components will also vary. Potteiger et al. show a great example of how an example rail-way system can be divided into components [43].

5.1.2 STRIDE

STRIDE is an acronym that stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. The method was first invented by Kohnfelder and Garg at Microsoft [46], where the overall goal of the method was to help developers identify potential attacks in order to design more secure systems. STRIDE is described in more detail in Table 5.1.

	Threat Category	**Property violated**	**Definition**
S	Spoofing	Authentication	Impersonating something or someone
T	Tampering	Integrity	Modifying data
R	Repudiation	Non-repudiation	Claiming that you did not do something
I	Information Disclosure	Confidentiality	Providing information to a non-authorized party
D	Denial of Service	Availability	Making service(s) unavailable
E	Elevation of Privilege	Authorization	Allowing someone to perform actions not normally permitted

Table 5.1: The STRIDE threat modeling method [42].

5.1.3 Component attack trees

Component attack trees, CATs, are constructed for each of the six STRIDE threat categories. In other words, each component will have six different attack trees, each of them illustrating different threat categories. An example of such an attack tree is shown in Figure 5.1. There are a few different nodes present in this tree: the root node, intermediary nodes, leaf nodes, and sub-tree nodes. The root node describes the corresponding STRIDE threat category and the component's name. Intermediary nodes and leaf nodes are slightly different - leaf nodes contain the direct threat whereas the intermediary nodes describe the *threat group*. The threat group is essentially seen as a node grouping together two or more threats, meaning that these threats are all attack vectors for the same threat group. The sub-tree node can be seen as a cross-reference to a different attack tree [47]. Successfully attacking

the root node in the referenced attack tree will allow an attacker to further advance their attack in the referring tree.

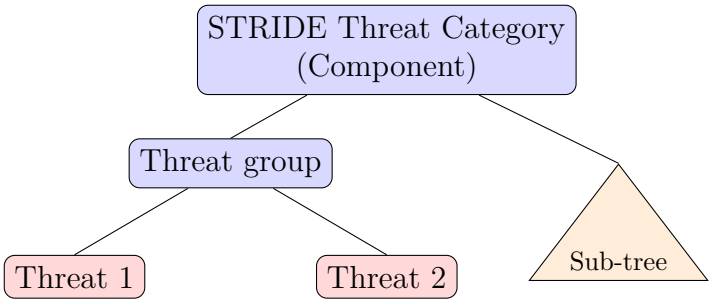

Figure 5.1: Example attack tree for a generic component.

An attacker starts at a leaf node and thereafter iterates upwards through an intermediary node path all the way to the root. When the attacker reaches the root, they have successfully breached the system via the threat category defined in the root node. An important detail is that each of the tree branches portrays logical OR relationships, meaning that either of the branches may be fulfilled in order to successfully fulfill the root node goal. In the following sections, all attack trees that are presented follow this pattern.

5.2 Threat severity scale

As mentioned at the beginning of the chapter, the QTMM method uses CVSS in order to classify the threats into different severity levels. However, some CVSS metrics are either out of scope or simply not applicable to electronic exams. Therefore, the threats in this chapter will be classified according to more suitable metrics.

The two CVSS metrics *attack complexity* and *user interaction* will be used. As defined in CVSS, the user interaction metric refers to a user other than the attacker, needing to participate in the attack for it to be successful [44]. For electronic exams, the attacker *is* the user, the cheating examinee. Therefore, the user interaction metric as defined by CVSS needs to be re-defined slightly. User interaction, in this case, refers to how much the examinee (the attacker) needs to interact with their computer in an anomalous fashion during the exam in order to perform the cheating attack. Anomalous interaction is defined as interaction that deviates from regular exam writing, which usually has a higher chance of being noticed by an invigilator. To make up for this re-definition, another metric called *third-party interaction* will be defined, which has the same meaning as user interaction does in CVSS. Thus, third-party interaction refers to the interaction from another person helping the examinee.

In addition to these three metrics, the threat's *portability* will be considered. The portability metric has earlier been introduced by Hietanen [3], during his work of identifying potential cheating threats. This metric measures the ease of porting an

exploit. Portability is considered to be high if it is possible to cheat with an obtained exploit *without* any manual work, such as making changes to the exploit.

A threat will be considered *severe* if the following holds:

- The *attack complexity* is low
- The *user interaction* is low
- The *third-party interaction* is low
- The *portability* is high

5.3 Attack tree completeness

The biggest difficulty when constructing attack trees is knowing when a tree is *complete*. The term complete here refers to whether or not a tree includes *the largest possible amount of attack vectors*. After all, in order to be able to protect a system from a specific attack, you must first realize that this type of attack is a threat in the first place. This is one of the more difficult factors when threat modeling, since how secure the system will be in the end largely depends on what threats you find throughout the modeling process. It would also be naive to assume that one is able to include all such threats and therefore be able to mitigate them all when designing a system. Another valid point is that there might exist new types of threats in the future, which are even harder to take into account when threat modeling today.

Sonderen [48] presents a "manual for attack trees" which describes the process of constructing attack trees for a system. He outlines two phases: *the discovery phase* and *the detailing phase*. The goal of the discovery phase is to discover as many different attacks as possible, which should result in a tree that contains at least the root and one layer of attacks. This can be done using various approaches, one of them being brainstorming. After the discovery phase is done, the detailing phase begins where the overall goal is to detail all the discovered attacks. According to Sonderen, when all the attacks are detailed enough, this results in a complete attack tree. Our main concern here is that the resulting trees from these two phases may differ depending on what actors are constructing them. Since the first phase includes approaches such as brainstorming, this suggests an approach similar to "try to think about every single possible attack vector within the system". This might be fairly easy for someone proficient in security but is most definitely harder for people with less experience. Thus, the initial question still remains: when is the tree complete, meaning that one has included the largest possible amount of attack vectors?

A large effort has been put into making the trees presented in this chapter as complete as possible, by including all possible threats that we could find. As discussed earlier in the chapter, this has been done both via brainstorming but also by reading previous work within the area. By choosing the QTMM threat modeling method and making use of STRIDE, we have managed to construct trees that are relevant to every threat category.

5.4 E-exam cheating threats

Using the threat modeling method QTMM described in Section 5.1, we have successfully been able to identify relevant threats for e-exams.

For an e-exam, there exist three main components: the LMS, the e-exam environment (also known as EEE), and the examinee. In order to construct CATs for each component, we first had to identify the relevant STRIDE threat categories, which can be seen in Table 5.2. Some threat categories do not apply to specific components, and some are simply out of this thesis' scope. Most of the threat categories are out of scope for the LMS component, since the focus of this thesis is not directly the LMS. Threats that directly affect the e-exam environment via the LMS are still of interest, since they may impact the security of the environment. For all three components, **D** (Denial of Service) is either out of scope or not applicable. Even though denial of service is a valid threat against the LMS, it is out of the scope of our work since we are focusing on securing e-exam environments. There are some cheating-related threats that could be considered relevant to the denial of service category, such as spamming network traffic to a critical component of the EEE. However we will not be taking these types of threats into account since these often require mitigations that rely on external solutions, such as denial of service protection.

	E-exam environment (EEE)	**LMS**	**Examinee**
S	✓	✓	✓
T	✓	Out of scope	Not applicable
R	✓	Out of scope	Not applicable
I	✓	Out of scope	✓
D	Not applicable	Out of scope	Not applicable
E	✓	Out of scope	✓

Table 5.2: Relevant STRIDE threat categories for each component.

5.4.1 E-exam environment

As seen in Table 5.2, all threat categories except for **D** are relevant for the e-exam environment (EEE) component. In order to fully understand the relevant CATs for these threat categories, we must first define what the threat categories mean in the EEE context.

Table 5.3 describes the relevant threat categories of an EEE in more detail. The *EEE property* specifically defines what an EEE must successfully do in order to prevent the threats in the corresponding STRIDE threat category. For spoofing, the underlying EEE property states that the EEE authenticates *correct device and software usage*. Using rather broad terms, correct device and software usage refers to using the correct, unmodified EEE software on a device that hasn't been specifically modified to give the examinee an advantage. If an examinee manages to circumvent this, this means that they have successfully managed to spoof their EEE. Similarly, for information disclosure, *confidential information* refers to information that would give

the examinee an advantage if they knew about it. Lastly, in elevation of privilege, an *authorized action* is one that is in line with the aim of the EEE. For a lock-down EEE, the aim is to prevent the examinee from accessing anything outside of the EEE. Any action that results in the examinee being able to access anything outside of the lock-down EEE is, therefore, an *unauthorized action*, and successfully elevates an examinee's privileges.

	EEE property	**Successful property violation**
S	EEE authenticates *correct device and software usage*	An examinee is successfully able to use the device and/or software incorrectly
T	EEE checks EEE integrity	An examinee is successfully able to alter the code and/or functionality of the EEE
R	EEE makes sure that any cheating-related activity is logged via logging	An examinee is successfully able to circumvent logging of certain activity
I	EEE makes sure that an examinee is not able to obtain *confidential information* related to the EEE	An examinee is successfully able to obtain confidential information about the EEE
E	EEE makes sure that every action performed by an examinee is *authorized*	An examinee is successfully able to perform an unauthorized action giving them an advantage

Table 5.3: The STRIDE threat categories for the EEE component, and their corresponding contextual properties.

EEE Spoofing The CAT for spoofing an EEE is shown in Figure 5.2. The tree contains various threats such as running the EEE inside of a virtual machine (**ES4**), injecting data into the EEE via a proxy (**ES1**), and remotely using the EEE (**ES3**). As for remotely using the EEE via third-party remote control software such as TeamViewer[1] or Zoom[2] (**ES3**), an examinee would have easy access to material outside of the EEE, while still being able to control the EEE via the application. This would also require the examinee to run the EEE via a remote computer, and then connect to the remote computer via another computer in the exam hall. Using the EEE outside of the exam hall (**ES2**) could also be a possibility, which means that the examinee would not even be present in the hall to begin with. Another possibility of spoofing would be to use another Wi-Fi network (**ES5**) – even though this doesn't necessarily mean that an examinee is able to access prohibited materials directly (in the case of a lock-down EEE), it still falls under the spoofing category since the examinee is not using their device correctly. The threat could also be combined with other threats: for instance, an examinee might have to use another Wi-Fi network in the first place in order to even be able to access prohibited material since the dedicated exam network might deny access to prohibited websites. Lastly,

[1] `https://teamviewer.com/en/`
[2] `https://zoom.us/`

tampering with the EEE in any way means that one is using the EEE incorrectly, which in turn means that the examinee is using a spoofed EEE. Therefore, the CAT includes a reference to the tampering sub-tree which can be seen in Figure 5.3.

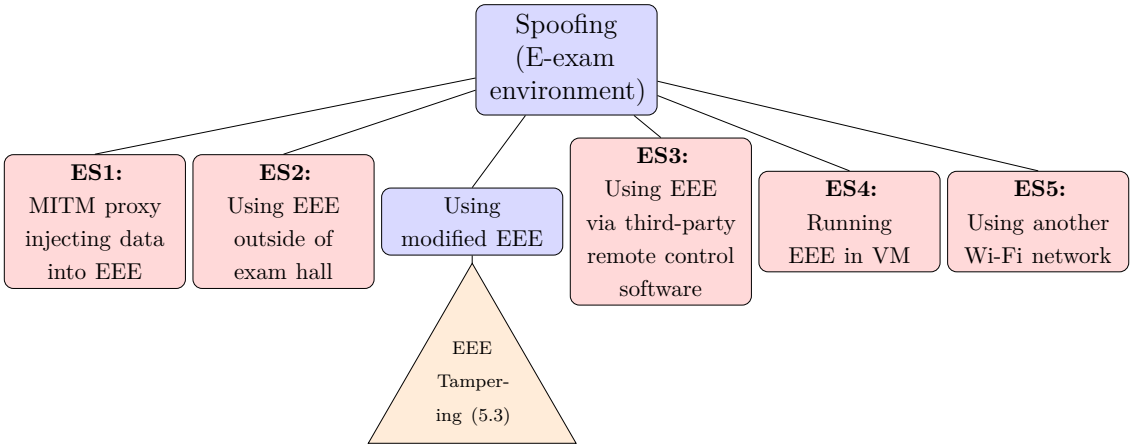

Figure 5.2: CAT for the *spoofing* STRIDE threat category of an e-exam environment.

EEE Tampering The tampering CAT for an EEE, which can be seen in Figure 5.3, includes several threats that an examinee could make use of in order to successfully tamper with the EEE. The tree contains three major categories that could be tampered with in order to alter the functionality of the EEE: the software, the operating system (OS), and the hardware. Some threats within these three categories are very specific for certain types of EEEs, and may not be applicable to all of them. For example, **ET9** (*redirect allowed third-party processes*) will only be a valid threat against EEEs that specifically allow third-party processes to be run in the first place.

EEE Repudiation In the case of an EEE, repudiation is about successfully being able to circumvent logging of a certain activity, as mentioned in Table 5.3. The CAT related to EEE repudiation is shown in Figure 5.4. There exist three ways for an examinee to circumvent logging: by disabling it completely, by preventing the logging from actually happening, or by faking the logging. Disabling or preventing logging is done by tampering with the EEE. The type of logging that an examinee may wish to circumvent for cheating purposes is also solely dependent on what the EEE actually is logging. For an examinee to successfully be able to cheat, it might be sufficient to only remove part of the logging, meaning that removing all logging may not be necessary. It may also be more suspicious if an examinee disables logging altogether since that means that the EEE would not be logging anything at all. Another approach is for the examinee to fake the logging (**ER1**), meaning that they are trying to remove the traces of their cheating or simply exchanging the suspicious logs with more acceptable ones.

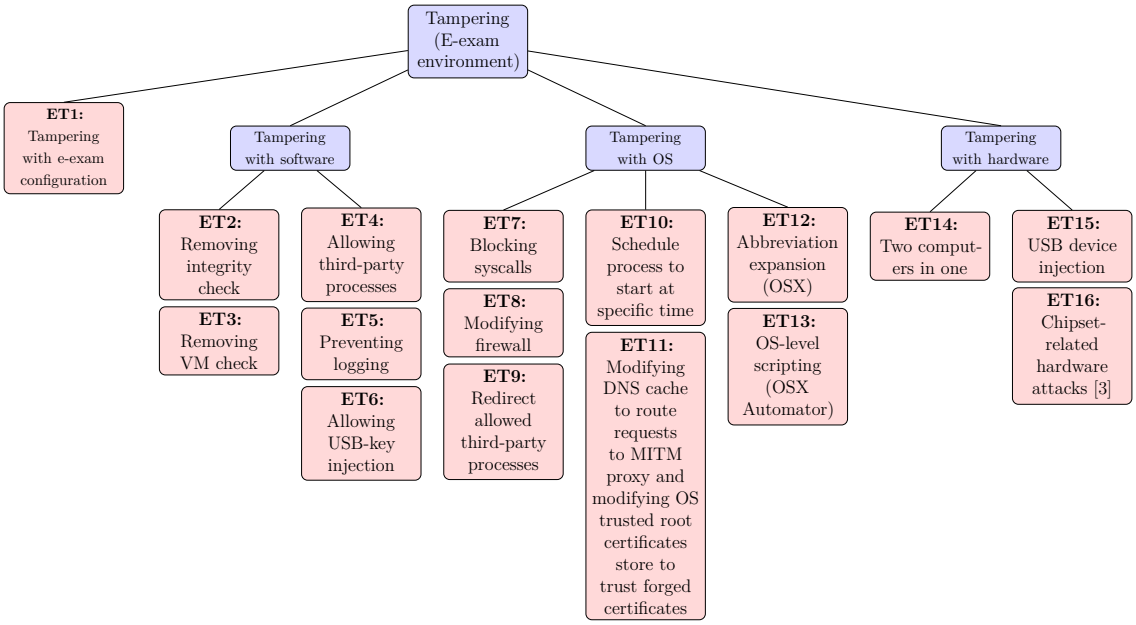

Figure 5.3: CAT for the *tampering* STRIDE threat category of an e-exam environment. Note that all child nodes that appear as columns are direct children of the first appearing intermediary group node.

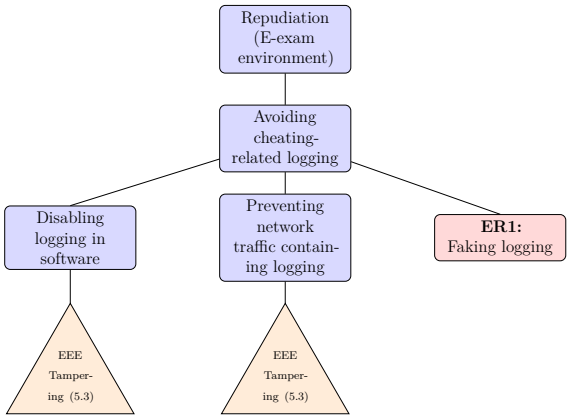

Figure 5.4: CAT for the *repudiation* STRIDE threat category of an e-exam environment.

EEE Information Disclosure For information disclosure, a cheating examinee's goal would be to obtain confidential information about the EEE, which in turn would give the examinee an advantage. The related threats are shown in the CAT in Figure 5.5. For example, for a closed-source EEE, a valid threat would be if an outside party manages to reveal parts of the source code via techniques such as reverse engineering or decompilation (**EI2** and **EI3**). By doing this, one would gain knowledge about the internals of the software, which in turn could potentially lead to an understanding of how to break the software. Similarly, for EEEs that allow defining specific third-party processes that are permitted throughout an exam, a valid threat would be if someone finds out about these processes before the exam

has begun (**EI1**). This imposes a new threat since an examinee could alter the functionality of the third-party software beforehand, in order to cheat. Lastly, retrieving confidential information from the configuration file is also a valid threat, if that is applicable to the EEE. In the case of encrypted configuration files, one can do this by brute-forcing the encryption key (**EI4**), or simply breaking the cryptographic scheme (**EI5**).

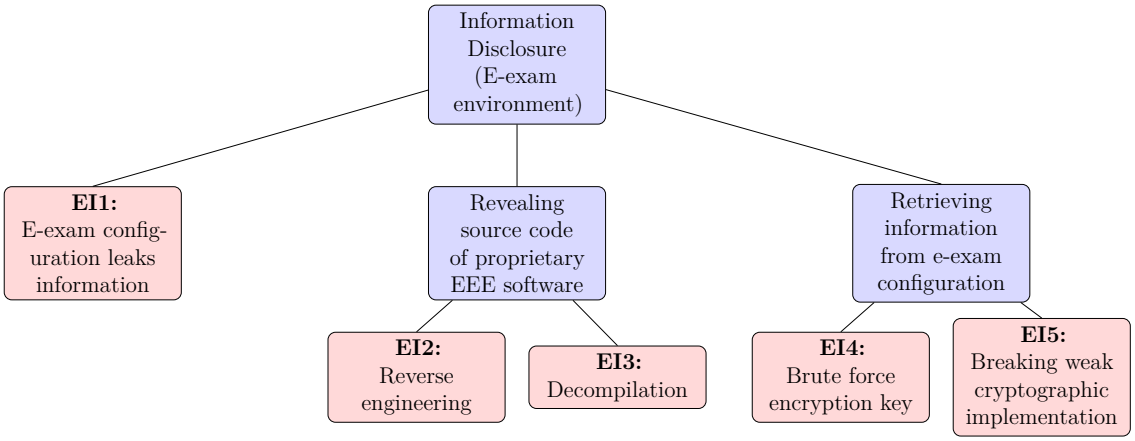

Figure 5.5: CAT for the *information disclosure* STRIDE threat category of an e-exam environment.

EEE Elevation of Privilege The last CAT for the EEE, related to elevation of privilege, can be seen in Figure 5.6. It includes three main categories: breaking out of the EEE, assistance/collaboration, and accessing forbidden materials. In order for an attacker to accomplish either of these, they need to either spoof or tamper. The reasoning behind this is simply because the concept of privilege elevation is a very broad topic, and therefore there are numerous approaches one can take in order to successfully elevate their privileges.

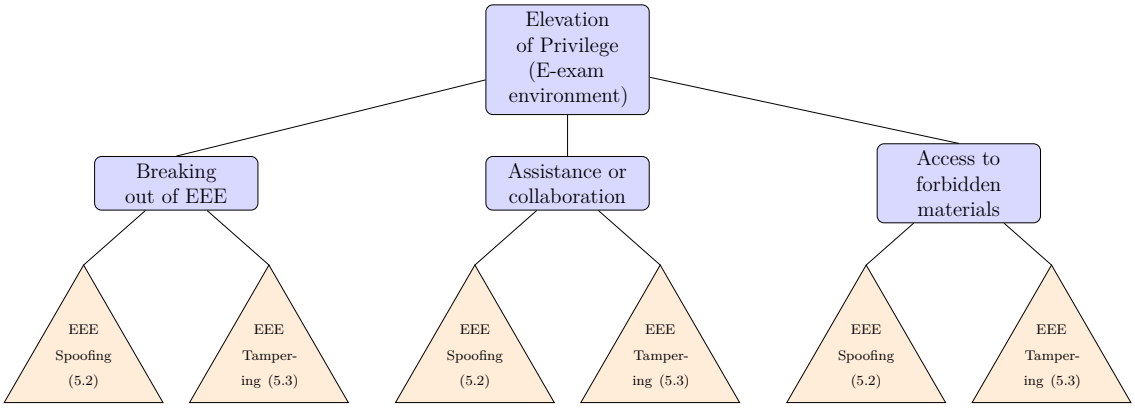

Figure 5.6: CAT for the *elevation of privilege* STRIDE threat category of an e-exam environment.

5.4.2 Learning Management System

For the LMS, only the spoofing threat category was found to be in scope and applicable. Successfully spoofing the LMS means that an examinee has successfully been granted access to an exam using an *incorrect EEE*. The term incorrect EEE refers to an EEE that deviates from the one that is specified to be used during an e-exam (most likely through the LMS).

LMS Spoofing The spoofing CAT for the LMS can be seen in Figure 5.7. There are two main ways an examinee could gain access to an exam using an incorrect EEE, either by using a completely *invalid EEE*, or *without* using the EEE at all (**LS2** and **LS3**). An invalid EEE could either be an older version (**LS1**), which deviates from the version that the examinee should use, or it could simply be a modified EEE. Modifying refers to tampering, thus, the tampering sub-tree from Figure 5.3 is included in the CAT.

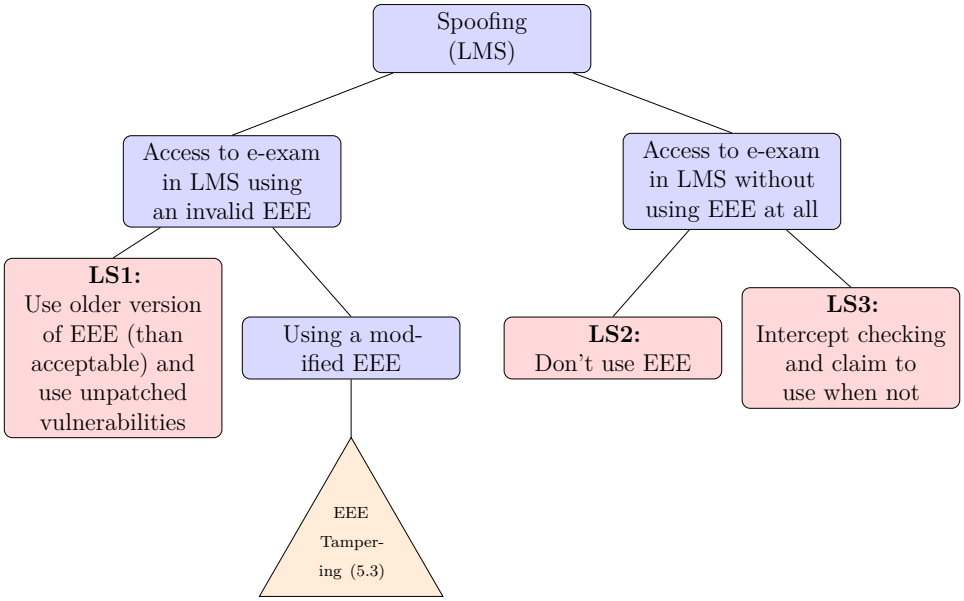

Figure 5.7: CAT for the *spoofing* STRIDE threat category of an LMS.

5.4.3 Examinee

For the examinee component, three relevant STRIDE threat categories were identified: Spoofing, Information Disclosure, and Elevation of Privilege. These three threat categories in the context of the examinee component are further described in Table 5.4. *Authorized information* related to the information disclosure category is defined as information that is purposefully disclosed by the examiner or any other similar authority to all examinees prior or during the e-exam. Any information that has not been purposefully disclosed by an authority to all examinees is considered unauthorized. Obtaining such information, therefore, results in information disclosure. Related to the elevation of privilege category, *authorized actions* refers to actions that are allowed during the examination session. Some of these actions likely

differ significantly depending on where the examination is to take place, universities might have different rules for toilet breaks, or for asking questions. However, it is likely that rules regarding forbidden materials are at least somewhat similar, so a common *unauthorized action* likely belongs to the group of actions that results in the examinee obtaining information that gives them an advantage. It is also worth mentioning that if an examinee successfully elevates their privileges in any way during the exam, this implies that they have successfully cheated.

	Examinee property	**Successful property violation**
S	Examinee authenticates identity	Examinee is successfully able to identify as another person
I	Examinee only obtains *authorized* information	Examinee is able to successfully obtain *unauthorized* information
E	Examinee only able to perform *authorized actions*	Examinee successfully able to perform an *unauthorized action* giving them an advantage, which in turn leads to successful cheating

Table 5.4: The STRIDE threat categories for the examinee component, and their corresponding contextual properties.

Examinee Spoofing The first CAT, related to spoofing, can be seen in Figure 5.8. Recall that spoofing for an examinee is defined as successfully identifying as another person, as presented in Table 5.4. We managed to identify three main threats: switching devices during the exam (**XS1**), using another examinee's account (**XS2**), and faking identification (**XS3**). Of course, some of these are more feasible than others, especially since we are looking at actively invigilated in-hall exams. For example, two examinees switching their devices during the exam might be unreasonable, since an invigilator would easily spot this. However, it is still considered to be a spoofing-related threat.

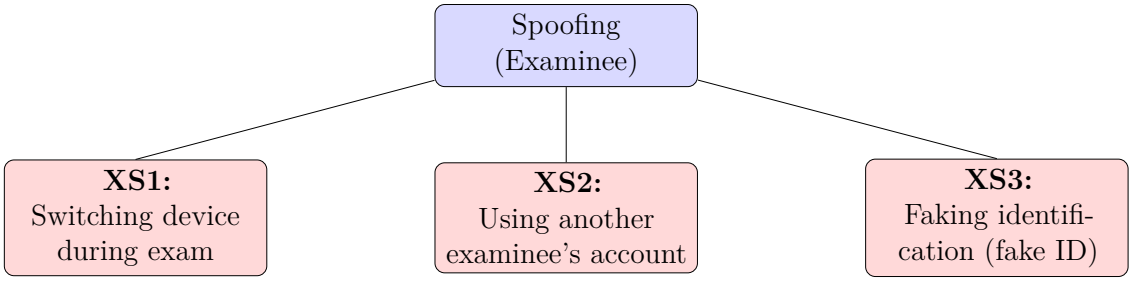

Figure 5.8: CAT for the *spoofing* STRIDE threat category of an examinee.

Examinee Information Disclosure The second CAT, related to information disclosure, is presented in Figure 5.9. As presented in Table 5.4, successful information disclosure for an examinee is about obtaining *unauthorized information*. This can be done by communicating either with another examinee, or an external third party. Two types of communication exist: either using the computer or by using

traditional methods (**XI5**) that are commonly used during paper-based exams as well. One example of such communication done via the computer is getting remote help via third-party communication channels (**XI1**), such as Discord[3] or Skype[4]. Another possibility would be communicating via side-channels, which includes Wi-Fi hotspot naming (**XI2**), Bluetooth device naming (**XI4**), and proxy information injection (**XI3**).

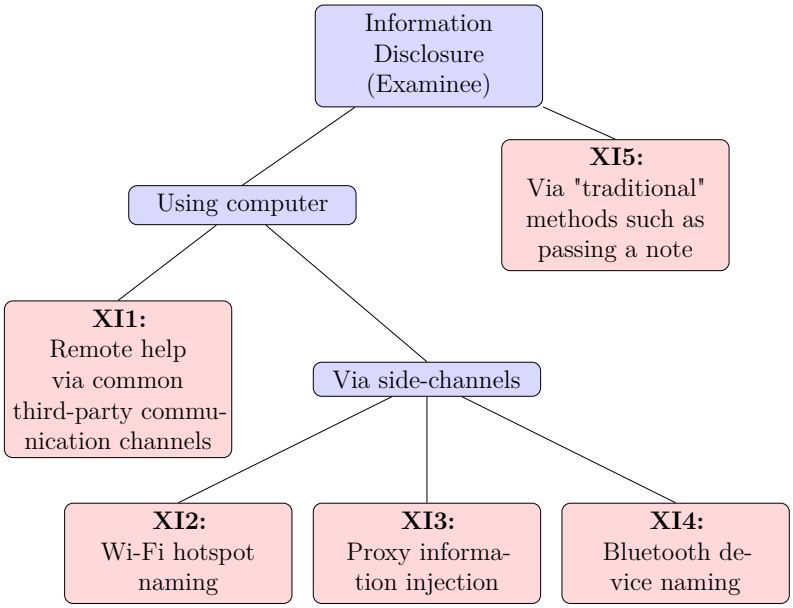

Figure 5.9: CAT for the *information disclosure* STRIDE threat category of an examinee.

Examinee Elevation of Privilege For the third and last threat category for the examinee component, elevation of privilege, there is no directly defined CAT. Recalling from Table 5.4, elevation of privilege refers to an examinee successfully being able to perform *any unauthorized action*. If an examinee is able to do this, it implies that they have cheated. Therefore, any of the previously presented attack trees for all of the components have chances of potentially leading to elevation of privilege for an examinee. This is further emphasized in Section 5.5, where it is shown that all other trees are connected to elevation of privilege of an examinee.

5.4.4 Threat severity

Table 5.5 summarizes all the threats presented in the attack trees above, along with their corresponding threat severity level. The table contains both the threat ID along with a description of each threat. In order to categorize each threat into its corresponding severity level, we have utilized the scaling presented in Section 5.2. A threat ID is marked with red if the threat is considered *severe*, meaning that the attack complexity (AC) is low, the user interaction (UI) is low, the third-party

[3] `https://discord.com/`
[4] `https://skype.com/en/`

interaction (TP) is low, and the portability (PA) is high. Additionally, a threat ID is marked with orange if the threat has three out of the four criteria for being severe. This kind of threat would not be considered *severe* but is still likely to be of concern.

ID	Threat Description	Metric severe			
		AC	**UI**	**TP**	**PA**
ES1	It might be possible to configure an EEE to connect to a proxy instead of directly connecting to an LMS, allowing an examinee to let the proxy inject data into responses that are shown inside of the environment.		✓	✓	✓
ES2	Misconfigurations might open up the possibility for an examinee to write their exam using the EEE outside of an exam hall, meaning that no invigilators can make sure that they aren't cheating.		✓	✓	✓
ES3	If the EEE allows the usage of third-party remote control software, an examinee can use two hosts to do their exam. One host runs the EEE, and the other host remotely controls the first. This requires that an accomplice agrees to help during an exam.	✓	✓		
ES4	An examinee can potentially launch a virtual machine on their host, running an entire operating system and the EEE inside of it. This might allow an examinee to use their host normally, circumventing the EEE completely.	✓	✓	✓	✓
ES5	Some cheating mitigation methods might be looking at the network traffic for a specific network dedicated towards e-examinations. However, unless correctly configured, an examinee might still be able to access their exam via a different network.	✓	✓	✓	✓
ET1	In cases where the EEE depends on some provided configuration, such as SEB (see Section 4.2), an examinee may be able to tamper with the configuration that the EEE loads and uses, giving the examinee complete control of the environment that is used. This could potentially lead to an examinee disabling certain aspects of the environment, allowing them to cheat.		✓	✓	✓

Table 5.5 continued from previous page

ID	Threat Description	AC	UI	TP	PA
ET2	Ensuring that an EEE launches without modifications could be done with integrity checks. Depending on the implementation of such an integrity check, an examinee might be able to either remove it completely or circumvent it.		✓	✓	✓
ET3	Similar to an integrity check, an EEE might refuse to launch if it is started inside a virtual machine. Depending on the implementation of this check, it might be circumventable or removable.		✓	✓	✓
ET4	Some EEE's might prevent an examinee from launching third-party applications at the same time as the EEE is being used. An examinee might be able to tamper with the EEE to remove this prevention mechanism and instead allow them to launch third-party applications.		✓	✓	✓
ET5	Suitable logging might be implemented in an EEE, but if an examinee is able to tamper with the environment, the logging might be preventable. This means that an examinee could remove the logging that would otherwise contain evidence of them cheating.		✓	✓	✓
ET6	Blocking key injection through peripheral devices might be blocked inside the environment, but successfully tampering with the environment might allow them to unblock it. This could lead to them being able to inject text through badusb devices such as a Flipper Zero[5].		✓	✓	✓
ET7	Tampering with the running operating system might allow an examinee to block certain *syscalls* that would otherwise be part of a cheating mitigation mechanism. This might allow them to cheat as the mitigation mechanism will not work as expected.		✓	✓	✓
ET8	It might be possible for an examinee to modify their firewall in such a way to prevent certain network traffic to and from their host. This traffic might be activity traffic that would otherwise catch cheating activity.		✓	✓	✓

[5] `https://flipperzero.one/`

Table 5.5 continued from previous page

ID	Threat Description	AC	UI	TP	PA
ET9	An EEE may implement functionality for allowing third-party applications inside the environment. This could allow an examinee to modify the third-party application or similar to have different behavior, or even be a completely different application. This could lead to an examinee having access to a different application than what was initially allowed, elevating their privileges and letting them cheat.	✓	✓	✓	✓
ET10	An EEE might only close disallowed processes as it is being launched, and then trust that its lock-down mode is sufficient to make sure that an examinee cannot launch other processes once inside the environment. This could make it possible for an examinee to launch disallowed processes simply by scheduling them to be started after a specified time after the environment has been started.	✓	✓	✓	✓
ET11	If the communication between an EEE and the currently used LMS uses insufficient authentication, then it might be possible for an examinee to exploit this fact. By modifying the DNS cache of the operating system and routing all traffic to an LMS through a proxy, the responses can be modified arbitrarily. This would let an examinee inject data into the responses that may have an effect on the availability of information in the environment, potentially letting an examinee cheat.		✓	✓	✓
ET12	OSX provides OS-level functionality for expanding pre-defined abbreviations. Examinees could potentially set up several fake abbreviations that expand to prohibited material in the form of text. In addition to this, OSX also facilitates *dictionary lookup* by simply selecting text, which in some cases looks up information on the internet for further context. This could allow an examinee to get information that it would not otherwise have access to.	✓	✓	✓	✓
ET13	OSX facilitates OS-level scripting through the OSX Automator. An examinee could potentially create sophisticated enough automations and scripts that could allow them to cheat.		✓	✓	✓

Table 5.5 continued from previous page

ID	Threat Description	AC	UI	TP	PA
ET14	An examinee could build their own, or acquire in some other way, a device that gives them access to two separate physical machines. A laptop with two discrete hardware setups inside of it, with the ability to switch between them, would completely circumvent the EEE.		✓	✓	
ET15	If an EEE does not block key injection from USB devices, it might be possible for an examinee to inject prohibited material in the form of text using a USB device, such as a Flipper Zero[6].	✓		✓	✓
ET16	An examinee may use platform-specific hardware features in order to attack the running operating system [3].		✓	✓	
ER1	An examinee might be able to fake parts of the logging solution in an EEE unless the EEE utilizes proper integrity validation of the logs. If an examinee can fake the activity logs, then an examinee will be able to either remove the traces of their cheating or simply exchange the suspicious entries with acceptable ones.		✓	✓	✓
EI1	An e-exam configuration file or similar could potentially leak information that could allow threats, such as **ET9**, to be executed with less effort.	✓	✓	✓	✓
EI2	In cases where an EEE is not open-source, or in some way relies on the fact that examinees do not know *how* the EEE works, an examinee can utilize reverse engineering techniques to eventually learn how it works. Upon learning *how* it works, several attacks may be possible, potentially allowing an examinee to cheat.		✓	✓	✓
EI3	An examinee can *decompile* the EEE, allowing them to much easier proceed with **EI2**.	✓	✓	✓	✓
EI4	Depending on the usage of encryption in the EEE, brute-forcing encryption keys or similar could be a viable option for an examinee to gain access to information that they would otherwise not have. This could be information that is specific to a configuration file, or even parts of an exam.		✓	✓	✓

[6] `https://flipperzero.one/`

Table 5.5 continued from previous page

ID	Threat Description	AC	UI	TP	PA
EI5	Similar to **EI4**, improper usage of cryptographic implementations may result in an examinee easily being able to bypass or completely break them. This could allow them to gain access to information that they would not otherwise have, potentially allowing them to cheat.		✓	✓	✓
LS1	Unless the usage of an EEE is properly verified, an examinee could potentially use an older version of the EEE and use unpatched vulnerabilities in the older version to their advantage.	✓	✓	✓	✓
LS2	Similar to **LS1**, it might be possible to simply *not* use the EEE if the usage of it is not properly verified, such as opening the exam through a regular browser.	✓	✓	✓	✓
LS3	In cases where an EEE provides some kind of authentication that it is being used, such communication may be interceptable. Depending on the implementation, an examinee could intercept the communication and provide the same authentication that the EEE would provide. This technique could be used in conjunction with **ET11**.		✓	✓	✓
XS1	For in-hall BYOD exams, two examinees could switch devices during the exam, allowing one examinee to help another. This would be similar to examinees switching papers during a paper-based exam.	✓			
XS2	Depending on how an examinee provides authentication to an LMS, it may be possible for examinees to share credentials amongst each other prior to an exam, and thus use another examinee's account. This would allow examinees to help each other in ways that would not be possible in a paper-based exam.	✓	✓		
XS3	Identification might be part of the authentication process of an examinee, which in-hall invigilators will be required to verify. Malicious invigilators or fake identification could be used to bypass such verification by examinees.		✓	✓	
XI1	Depending on the EEE, it might be possible to use common third-party communication channels. This would allow an examinee to communicate with others, allowing them to cheat.	✓			

Table 5.5 continued from previous page

ID	Threat Description	AC	UI	TP	PA
XI2	With access to view the different Wi-Fi networks available to connect to, an accomplice could create Wi-Fi hotspots outside of an exam hall and name them in ways that provide information to the examinee.	✓			
XI3	Similar to **ET11**, a proxy can be used to inject information that is provided by an accomplice or other examinees. This would allow an examinee to cheat.		✓		✓
XI4	Similar to **XI1**, an accomplice can name Bluetooth devices to information that is advantageous to an examinee. These names could then be viewed in a device's Bluetooth connection settings.	✓			
XI5	Finally, traditional methods such as passing paper notes or communicating vocally inside an exam hall are threats as well.	✓			

Table 5.5: Table summarizing every threat in the attack trees presented in Chapter 5 along with their severity levels.

5.4.5 Previous findings

Some of the threats presented above have previously been shown in earlier work. Hietanen [3] presents threats related to different types of e-exams: controlling workstation exams, controlling operating system exams, and controlling software exams. The threats presented by Hietanen related to controlling operating system exams are far more in-depth than the ones presented in the attack trees above. An example of a more complex attack that Hietanen shows is the *PCI-related hardware attack*. This attack is an example of a very specific low-level attack, related to accessing the memory space of the e-exam OS through the usage of Direct Memory Access and a PCI device. As Hietanen mentions in his thesis, these types of attacks are *very* challenging to implement, meaning that the average user would most likely not be able to cheat via these types of attacks. Some threats in the attack trees have been left relatively vague in order to more easily be able to create more general design principles, which are presented in Chapter 6. The hope is that the general design principles also cover the more specific attacks, such as those that Hietanen presents. Such general design principles should be favored so as to cover as many possible specific attacks while keeping the number of principles low.

Sindre et. al [1] and Küppers et. al [22] compare paper exams and e-exams, examining what additional threats e-exams introduce in addition to the threats that already exist for paper exams. Some of the threats presented in this chapter have been shown in these papers, using different threat modeling techniques.

5.5 Cheating threat relationships

The threat categories of STRIDE all correlate to each other, in different ways. In other words, if an attacker successfully manages to attack a system where the attack falls into some certain threat category, this very often leads to subsequent threats on the system as a result from the first attack. For example, if an attacker successfully manages to *tamper* with the system, this most certainly leads to *elevation of privilege*. Similarly, *tampering* with something may also result in *spoofing*. The point is that each of the categories cannot be seen as independent of one another: in reality, they are dependent.

The relationships between the CATs previously constructed can be illustrated through a state diagram, which can be seen in Figure 5.10. As can be seen in the figure, all states eventually lead to *elevation of privilege* for an examinee. This is because the definition of elevation of privilege from the examinee perspective is cheating. If an examinee manages to elevate their privileges in any way throughout an exam, they have successfully cheated. The relationships displayed in the figure can also be seen in the corresponding CATs. For example, the spoofing CAT for an LMS seen in Figure 5.7 includes an EEE Tampering sub-tree. Therefore, EEE tampering is connected to LMS spoofing, shown in Figure 5.10.

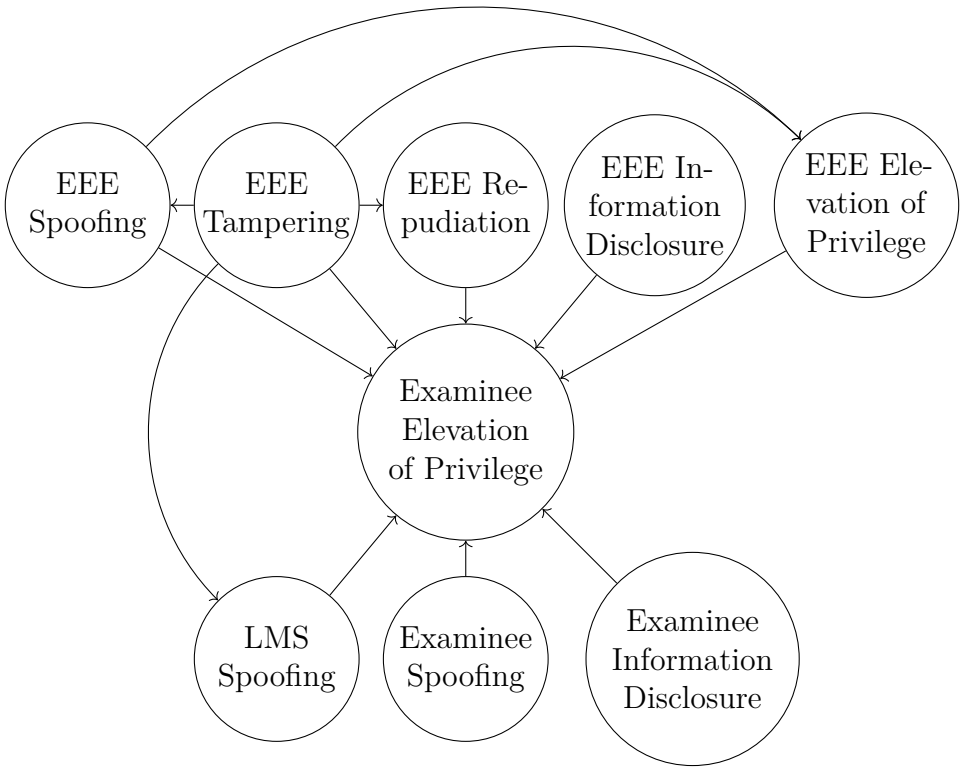

Figure 5.10: State diagram illustrating the connections between the different CATs for LMS, Examinee, and EEE.

6

Secure design principles

Security should be part of the design process. Not only does it minimize the number of potential vulnerabilities early on, but it also minimizes the cost that has to be paid later on in the form of technical debt. The concept of including security in the design process is nothing new, security by design is a common approach to securing software [4].

In this chapter, we propose *design principles* used to design secure e-exam environments. Similar principles have been proposed in other areas: Prudent Engineering Practice for Cryptographic Protocols by Abadi and Needham [5] is an example of such. Our design principles are constructed around six pillars: trustworthiness, fairness, supervision, transparency, scalability, and usability. Figure 6.1 shows a visual representation of the pillars. The pillars were carefully selected after the thorough threat modeling process in Chapter 5, where we identified severe threats against e-exam environments in a BYOD setting.

An overlying goal of the principles has been to make them as general as possible, while still making sure they cover a large number of possible vulnerabilities. As previously described in Chapter 3, there are different types of e-exam environments that differ in design, which is why the principles need to be general in order to hopefully suit all types of e-exam environments.

Finally, one of the most challenging aspects of designing such environments used in a BYOD setting is the fact that we are trying to restrict the user from their own machine. This means that the user has complete and full access to their machine, both physically and at an operating system level. The general view is that a BYOD e-exam will never be as secure as a non-BYOD e-exam, because of the access level a user has. However, an attempt must still be made to find ways in which a BYOD e-exam can achieve at least a reasonable level of security. Reasonable meaning that although it might be less secure than a non-BYOD e-exam, a user would be required to exert *significant effort* to cheat successfully. The design principles presented in this chapter are part of such an attempt.

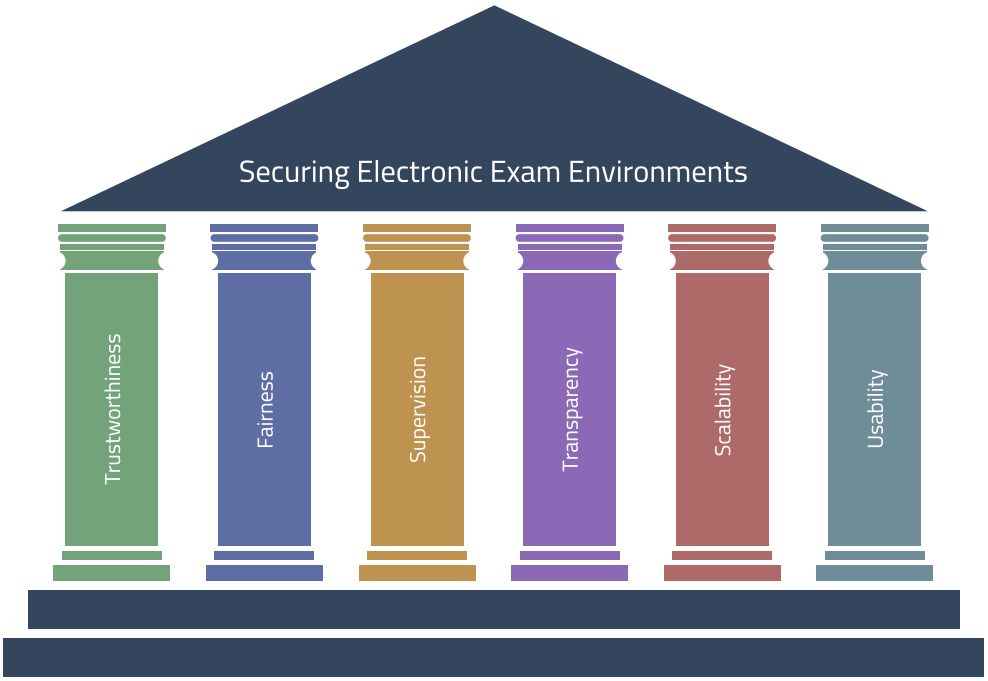

Figure 6.1: Visual representation of the design principle pillars.

6.1 Trustworthiness

The first pillar of designing a secure EEE is the *trustworthiness*. For this pillar we need to establish a type of trust model between our three components identified earlier in Chapter 5: the examinee, the e-exam environment, and the LMS. The first thing to realize is that an LMS must always assume that an examinee could be malicious and therefore never explicitly trust the examinee, similar to the zero-trust model [49]. The zero-trust model is a security concept that assumes that no user, device, or application can be trusted by default – very similar to the fact that the LMS should always assume that an examinee is malicious by default. By assuming that an examinee could be malicious, the LMS must always be certain that an examinee is truly using a trusted e-exam environment. If the LMS is uncertain about an examinee's usage of the e-exam environment at any point, then the e-exam environment has not provided enough verifiable proof that it is being used. This brings us to our first principle for secure e-exam environments.

Principle 1

> An e-exam environment *must* be able to convincingly prove (to an LMS) that an examinee is using a trusted environment.

For an EEE to *convincingly prove* that the examinee is using a trusted environment, it must be infeasible for a malicious examinee to prove this outside of the environment. A *trusted environment* is one that an LMS expects an examinee to be using during an exam. The trusted environment is therefore likely a specific version, or

variant, of an environment. For example, the attack tree found in Figure 5.2 contains various spoofing threats against an e-exam environment that would lead to an examinee using an untrusted e-exam environment. If an EEE cannot convincingly prove its usage, it would imply that the EEE contains fundamental design flaws that would allow a malicious examinee to either bypass the whole EEE or modify the EEE to their own liking. This would in turn make cheating very easy: it does not matter how well you secure the environment if one can bypass it completely by design.

As seen in Chapter 3, we present two types of e-exam environments along with their properties. The chapter includes examples of certain e-exam environments along with descriptions of how they authenticate that an examinee is using a trusted environment. For example, SEB utilizes specific HTTP headers appended to the network traffic originating from the application, as seen in Subsection 3.2.1 and Section 4.5. If an examinee could replicate these headers, it would imply that they could modify the environment to their liking, as long as they replicate the trusted headers. It could even go as far as them being able to bypass the whole application.

An important detail to realize is that anything the e-exam environment can do, the examinee can do the same. This brings us to the conclusion that an e-exam environment's security cannot rely on practices that an examinee can trivially replicate. In the case of utilizing HTTP headers, as SEB does, this implies that the calculation of the headers must not rely on insecure cryptographic practices such as insecure hash functions or insecurely stored cryptographic keys. If the authentication relies on cryptographic keys, the keys need to be stored securely so that an examinee cannot retrieve them in order to prevent replication of the authentication to the LMS. This decision is in line with the requirement v.4.0.3-2.9.1 of the OWASP Application Security Verification Standard (ASVS) [50], which explicitly states "Verify that cryptographic keys used in verification are stored securely and protected against disclosure, such as using a Trusted Platform Module (TPM) or Hardware Security Module (HSM), or an OS service that can use this secure storage".

The FLEX framework, as seen in Subsection 3.2.3, utilizes Remote Attestation (RA) in order to prove the trusted environment. Remote Attestation, related to Trusted Computing, has previously been shown to be an effective way of actively proving that a user's computer is in a specific trusted state. Balfe and Mohammed [51] show how to utilize Trusted Computing in order to mitigate cheating-related threats related to platform modifications for gaming consoles. The trusted state can be interpreted as a trusted environment, meaning that the trusted state is when the device has entered the *trusted, unmodified* e-exam environment. A Remote Attestation approach that is in line with the requirement v.4.0.3-2.9.1 of the OWASP ASVS mentioned above would utilize a securely stored key inside a Trusted Platform Module to sign the host's state, allowing a verifier to be confident that a host is in its trusted state.

Not only does the e-exam environment need to prove its current trusted state, but the security also relies on whether the LMS actively verifies the security measures taken. This brings us to our second principle.

Principle 2

> An e-exam environment *must* be certain that the LMS has verified the provided security measures.

The LMS should verify the security measures provided by the EEE, but if such verification is not enforced, developers integrating the EEE to be used with an LMS might make the integration insecure. If developers mistakenly integrate the usage of an EEE into their LMS without performing proper verification, examinees might be able to cheat. An example of such an oversight can be seen in Section 7.5, where an LMS does not perform proper verification, allowing an attacker full access to an exam outside the EEE.

The communication between an e-exam environment and an LMS is essentially a two-way street: both ways need to be secured in order to stop cheating. This means that the e-exam environment must be designed in such a way that an LMS has to verify the security measures provided. Otherwise, the LMS becomes the weakest link by not verifying the security measures, and an examinee may be able to bypass the e-exam environment as a result.

6.2 Fairness

The second pillar of designing a secure EEE is the *fairness*. An e-exam includes multiple parts that must be ensured to be fair, and the main point is that no examinee should have an upper hand regardless of what computational power the examinee has access to.

An examinee having access to prohibited materials during an exam gives that examinee an unfair advantage in comparison to other examinees. In paper-based exams, invigilators commonly make sure that no examinee has brought a cheat sheet or similar. For BYOD in-hall e-exams, the scenario is drastically different. For an invigilator to notice that an examinee has access to prohibited materials, the EEE could log all activity on the host computer, allowing for retrospective auditing of material access. It could also attempt to actively prevent an examinee from accessing such prohibited material in the first place, by blocking certain actions and utilizing a lock-down environment as discussed in Chapter 3. The third principle revolves around the fact that an exam should be fair for all examinees, meaning that nobody should have elevated privileges to access prohibited materials.

Principle 3

> An e-exam environment *must* give all examinees the same level of access to materials.

The principle revolves around the fact that it should be *hard* for an examinee to access prohibited materials. As mentioned earlier in our work, 100% security is generally perceived as an impossible goal. However, the principle states that all examinees should have the same level of access to materials, meaning that performing

some exploit to gain access to prohibited materials should require extreme efforts for an examinee. A well-implemented lock-down environment might still contain vulnerabilities, but exploiting them should require significant effort and they should ideally be the only way to break the environment. This is similar to how something is considered secure in the world of cryptography, where the most efficient attack is brute force. A similar approach should be applied to e-exam environments.

Another point this principle considers is compatibility between different operating systems: if the e-exam environment is software-based and can be run using multiple operating systems, no operating system should give an examinee the upper hand. If there would exist an operating system that would imply increased access for an examinee, the examinee would not have the same level of access to materials as others using a different operating system. This also implies that an e-exam environment must be designed in such a way that it works similarly on all accepted platforms, to prevent giving some examinees an upper hand.

6.3 Supervision

Supervision is the third pillar of designing a secure EEE. In contrast to paper-based exams, it is more challenging for an invigilator to monitor e-exams. Since we are moving from writing the exam on a device rather than on a paper, this also implies that an examinee by default has access to more material than they do in the paper-based exam. It would be naive to assume that an invigilator would be able to monitor every single screen in an exam-hall since there could potentially be hundreds of students taking an exam simultaneously. In paper-based exams, the invigilator can monitor the students with their eyes, but one unfortunately cannot scan a computer with their eyes. Therefore, the EEE needs to introduce a technical solution to this problem, meaning that it must utilize suitable logging for detecting cheating-related activity.

Principle 4

An e-exam environment *must* implement suitable logging.

The most challenging part of the principle is developing a solution that fits regional rules, such as those that come with GDPR [27]. This introduces a rather challenging constraint in terms of what information an EEE may log, since some information may be considered personal and therefore out-of-scope GDPR-wise.

As Principle 3 mentions in Section 6.2, all examinees must have the same level of access to materials. Like previously discussed, it is hard to ensure 100% security in any system, which also holds for e-exam environments. However, we can make it harder for an attacker to successfully breach the system by utilizing so-called *defense-in-depth* techniques. This means that even if an examinee would successfully gain access to any prohibited material, the e-exam environment must have suitable logging in place in order to detect such activity, as Principle 4 mentions. The same goes for any unauthorized action: it should not go unnoticed whenever an examinee

successfully performs one.

Principle 5

> An e-exam environment *must* notice when an examinee performs an unauthorized action.

The fifth principle can essentially be seen as a subset of Principle 4, which revolves around logging. In order to fulfill the principle, the e-exam environment could utilize logging that notices whenever an examinee is attempting to perform any *unauthorized action*. Similar to in Subsection 5.4.1, an unauthorized action is defined as an action that is not in line with the aim of the EEE, which means that any action that results in an examinee gaining access to anything outside of the EEE is unauthorized. An example of such an action would be if an examinee is accessing or even attempting to access prohibited materials.

The type of activity logged could potentially be DNS requests to see whether an examinee is accessing prohibited websites, monitoring the started processes on the device, and monitoring input/output from the system. The same must also hold for examinees that somehow manage to quit the e-exam environment before turning in their exam: since they are performing an unauthorized action, it must be flagged. If an examinee would be able to quit the e-exam environment unnoticed, they could access prohibited materials and then re-open the environment. Therefore, there must exist security measures in place that ensures that whenever an examinee exits the e-exam environment, the activity is logged. Perhaps the best way of mitigating cheating completely would be to prohibit examinees from quitting the environment until the exam has been turned in, but since computers are not foolproof, unforeseeable things could happen throughout the exam requiring an examinee to quit the environment. It is also unfeasible for an e-exam environment to prevent an examinee from turning off their device. Therefore, even if an examinee can potentially quit the environment, this should be noticed via logging or similar techniques.

6.4 Transparency

The fourth pillar, *Transparency*, is about the fact that good security in an EEE often comes from developing it in a transparent manner, which can be done by making a project open-source.

E-exam environments are not alone in this matter: previously it has been shown that software generally benefits from being developed using a transparent approach. An example is "The Digital Exam Monitor" (originally "Den Digitale Prøvevagt" in Danish) which was introduced in Denmark in 2019 to be used during e-exams, in order to monitor examinees' devices to mitigate cheating-related threats. The software was developed closed-source using obfuscation techniques. A high-school student reverse-engineered the software resulting in major security flaws being discovered, such as the possibility of bypassing the software completely. Furthermore, he also discovered that the software utilized insecure DES encryption in order to

hide a secret API key within the program used to store data from the examinees [52, 53].

This incident shows that relying on security through obscurity is not a reliable way of ensuring security, since it is only a matter of time until someone is able to reverse-engineer the software and discover the secrets underneath. Instead, software should be developed in an open-source manner, such that even if someone knows everything about the inner workings of the software, they still cannot break it. This brings us to the sixth principle of building a secure EEE.

Principle 6

An e-exam environment *must not* rely on security through obscurity.

The principle implies that developers of e-exam environments should choose an open-source approach over a proprietary closed-source one. Not only does this have a positive impact on the security of the EEE: but it also means that anyone can inspect the code and see what it actually does on an examinee's device. As seen in Chapter 3, we listed multiple EEEs in use today, one of which was Digiexam [19]. Digiexam is developed closed-source, meaning that examinees have no opportunity to inspect the software and see what type of information it logs about their device. Since it is closed-source, it could technically be logging stuff that is against GDPR [27] in many regions, even though the developers are claiming not to do so.

The open-source approach also comes with downsides. A cheating examinee will definitely have an easier time discovering vulnerabilities in an open-source EEE since the code is freely available to view. However, we believe that the advantages will outweigh the disadvantages in the long term, compared to the short term. It may be easier for malicious examinees to discover vulnerabilities in the short term, but once enough people know about them, someone will eventually report them to the developers resulting in them being patched. Honest users, such as institutions using the EEE, will also have an easier time discovering such vulnerabilities and can therefore contribute towards securing the EEE.

6.5 Scalability

The fifth pillar considers the *Scalability* of an EEE. Scalability has been shown to be one of the most important factors when developing an EEE [1] since e-exams could potentially be taken by hundreds of examinees simultaneously in the same exam hall. Therefore, it is important that an EEE is *scalable*, meaning that it should be easily available to scale to support a large number of examinees.

Principle 7

The security measures taken in the design of the e-exam environment *must not* extensively affect the scalability of the environment.

The principle states that the security should not *extensively* affect the scalability of the environment, meaning that it should be easy to conduct exams for a large number of examinees. The BYOD approach itself makes a positive impact on the scalability aspect since each examinee can bring their own device. However, there exists a difference between the two types of environments in terms of scalability as presented in Chapter 3, since OS-based e-exam environments usually require USB sticks to be handed out before the exam. A more detailed discussion regarding scalability and its importance is presented in Chapter 9.

6.6 Usability

The sixth, and last, pillar is *Usability*. Along with scalability, usability has also been shown to be among the most important aspects for an EEE [7]. The pillar highlights the fact that an EEE needs to be usable, despite the security measures taken in order to secure the environment. This concept is not new: the tradeoff between security and usability has been discussed in the past as well. It is important to make a system usable and secure, since security measures that impair the usability may cause users to resort to insecure behavior instead [54].

Principle 8

> The security measures taken in the design of the e-exam environment *must not* extensively affect the usability of the environment.

Since a tradeoff exists between security and usability, it would be naive to assume that we can design an EEE in the most secure way that does not affect usability. Instead, the principle states that the security should not *extensively* affect the usability of the environment, meaning that the EEE should still be easy for an examinee to use without requiring further assistance.

Sasse et. al. [55] discuss the *security-usability tradeoff myth* centered around the fact that developers often introduce too many unnecessary security measures resulting in lower usability. The goal is of course to harden the EEE until it requires extreme effort from an examinee to cheat, but we do not have to introduce security measures that are unnecessary, such as those which have not been proven to increase security.

6.7 Threat mitigation

With these presented design principles, we also show which threats they help mitigate in Table 6.1. It is evident that the design principles make sure that a wide range of threats are mitigated, since only four threats, ET14, XS1, XS3, and XI5 are left unmitigated. This is due to the fact that those four threats are purely physical threats that only an invigilator would be able to detect. ET14 is a threat that cannot be detected by an EEE, as an invigilator would need to notice that an examinee's device has two separate physical machines inside of it. For XS1, it would be impossible for an EEE to detect if two examinees suddenly switch devices. XS3

is similar in that there is no way for an EEE to discover that an examinee has falsely identified themselves in the exam hall. Lastly, XI5 are traditional cheating methods that exist purely in the physical world, making them impossible to mitigate with an EEE. Since only unmitigatable threats are left, we can argue that the eight proposed principles are enough.

Let us take ET11 as an example to clarify how we evaluated which principles that help mitigate the threat, shown in Table 6.1. As a reminder, ET11 is about the possibility that an examinee can utilize a man-in-the-middle proxy which all traffic between the e-exam environment and the LMS goes through. This proxy could, without proper authentication, inject data into the responses from the LMS which could have an effect on the availability of the information in the environment, potentially letting an examinee cheat. Also, for simplicity in this example, we will use the abbreviation P1 when referring to principle 1, P2 for principle 2, and so on.

If P1 is followed, then the LMS can be confident that the examinee is using a trusted environment, which would have to include proper DNS configurations. Without including DNS configurations as part of the trusted environment proof, an examinee can freely modify them without the LMS knowing. This would leave ET11 unmitigated, but with DNS configurations as part of the trusted environment proof, the DNS configurations can be trusted, mitigating ET11. P2 follows from P1 as the proof provided as part of P1 is worth nothing without P2. This is also why no threat can be mitigated by only one of the two; they have a symbiotic relationship.

P3 is different since it is about the fact that it should be difficult to access prohibited materials. As the injected data by the proxy can be considered prohibited material, it should require extreme efforts to successfully have a proxy inject data into the LMS responses. As P1 showed, we needed to include the DNS configurations as part of the trusted environment proof. To follow P3, we must therefore make sure that the forging of a valid proof requires extreme effort, since that would mean it requires extreme efforts to be able to inject data into the LMS responses. For P1 to be valid, this forging would also need to be difficult, but P3 emphasizes the significance of the forging difficulty. One could make sure that the proof is cryptographically hard to forge, which would make it require extreme efforts to forge, mitigating ET11.

For P4, proper logging of the received responses and perhaps a signature of their contents could be used to retrospectively notice the usage of a proxy. Logging that reveals DNS configurations could also be used to notice such usage. This would mitigate ET11. P5 is very similar and further emphasizes the significance of proper logging implementations.

The last principle, P6, is more or less about the development lifecycle of the e-exam environment, and less about implementation. Therefore, P6 cannot directly mitigate ET11, even if it has the potential to be a factor in the addressing and fixing of vulnerabilities related to ET11.

It is also evident from the table that P7 and P8 do not mitigate any of the presented cheating threats. This is due to the fact that these two principles are not meant to mitigate any threats, but they are still important in order for the design to take the

scalability and usability into account. Therefore, even though these two principles do not necessarily increase the security of an e-exam environment, they cannot be excluded.

Finally, many threats are covered by more than one principle, giving a *defense-in-depth* approach. This makes sure that to execute some exploit for a threat, several barriers of defense need to be broken down in order to successfully cheat.

In a perfect world, where an e-exam environment would be designed to adhere to all of the principles presented in the chapter, it would mitigate the found cheating threats and therefore could be technically considered as secure. However, as we have previously discussed, no system can be made 100% secure since new types of flaws may exist in the future. It may also be that more threats exist for e-exams environments, which we did not find during our threat modeling in Chapter 5.

	Principle helps mitigate							
Threat ID	1	2	3	4	5	6	7	8
ES1	✓	✓	✓	✓	✓			
ES2			✓	✓				
ES3			✓					
ES4	✓	✓	✓	✓	✓			
ES5	✓	✓	✓		✓			
ET1	✓	✓			✓			
ET2	✓	✓			✓			
ET3	✓	✓			✓			
ET4	✓	✓			✓			
ET5	✓	✓						
ET6	✓	✓	✓	✓				
ET7	✓	✓			✓			
ET8	✓	✓			✓			
ET9			✓	✓	✓			
ET10			✓	✓	✓			
ET11	✓	✓	✓	✓	✓			
ET12			✓	✓	✓			
ET13			✓	✓	✓			
ET14								
ET15			✓	✓				
ET16			✓					
ER1	✓	✓						
EI1			✓					
EI2						✓		
EI3						✓		
EI4						✓		
EI5						✓		
LS1	✓	✓		✓	✓			
LS2	✓	✓						
LS3	✓	✓						
XS1								
XS2				✓	✓			
XS3								
XI1			✓	✓	✓			
XI2			✓		✓			
XI3	✓	✓			✓			
XI4			✓		✓			
XI5								

Table 6.1: Summary of which principles help mitigate the found threats in Chapter 5.

7

Security analysis of Safe Exam Browser: A case study

This chapter aims to describe the case study that has been done on the e-exam solution used at Chalmers University of Technology. As described in earlier chapters, the e-exam solution used at Chalmers is the Safe Exam Browser (SEB) lock-down browser together with Inspera LMS. The primary focus of this study has been investigating SEB itself, but an additional point of interest has been the connection between SEB and Inspera. Similar to Chapter 5, the threats of interest have been ones that are relevant to the *during* phase of an exam. This does not imply that all relevant threats must be constructed throughout the *during* phase, in fact, there exist relevant threats that are prepared in the *before* phase which are executed in the *during* phase. The component attack trees (CATs) constructed in Chapter 5 have been used in order to identify potential threats in the solution. This implies that all attempted attacks are present in the corresponding CATs.

Additionally, we would like to mention once again that all testing has been done in communication with the exam administration at Chalmers. Our contact at the administration helped us gain access to the corresponding systems. Furthermore, we have made sure to report the found exploits to the administration. The aim of the case study has not been to actively help any student cheat, instead, the goal has been to identify and highlight the potential threats present in the e-exam solution at Chalmers.

7.1 Setup

This section will aim to describe the setup of the e-exam solution used at Chalmers University of Technology.

7.1.1 Safe Exam Browser

The versions of Safe Exam Browser (SEB) that have been used throughout this case study have been version 3.4.1 for Windows, and version 2.3.2 for MacOS, unless otherwise stated. The reasoning behind this is simply because these are the SEB versions that Inspera provides to the examinee, through a download link. The version numbers differ because them being isolated projects with their own versioning.

The case study has been performed using a gray-box testing approach, which is simply a combination of white-box testing and black-box testing [56]. Since SEB is open-source, meaning that it is freely available on GitHub[1,2], it was possible to inspect the code that SEB runs and therefore try to use that as an advantage. Additionally, the SEB developer documentation [57] has been studied whenever needed.

7.1.2 Inspera

In collaboration with the exam administration at Chalmers, we got access to the Chalmers Inspera administration panel. Due to ethical concerns, it has been important to not access any sensitive data. All tests have been performed using our own accounts, which have had specific roles assigned in order to prevent us from viewing any other exams except for those created by ourselves. Therefore, the exploits found in this chapter have not given us access to any sensitive information.

Using the administrator panel, we were able to create new exams which allowed us to perform testing. Figure 7.1 shows a screenshot of all the available SEB options that could be configured for a specific exam. The options correspond to the same options presented earlier in Table 4.1. *Invigilator password* is the same as *Administrator password*, and the *SEB password* is the encryption key used to encrypt the config file. As can be seen from the screenshot, only a few options are configurable. This does not mean that the SEB configuration file generated by Inspera only contains the options seen in the screenshot. Every configuration file contains some default options that are always enabled for every exam.

Inspera also allows examinators to monitor the examinees throughout the exam, by visiting the *Monitor* tab of the administrator panel. However, the monitor capabilities are pretty limited. The monitor tab contains a list of all the examinees registered for an exam and allows one to see what state they currently are in. An examinee's client may also generate warnings in case anything suspicious happens on their end. Examples of such suspicious things are if the examinee exits the exam and re-opens it again later, or if the examinee for some reason has network problems. Figure 7.2 shows an example of warnings generated for an examinee.

[1] `https://github.com/SafeExamBrowser/seb-win-refactoring`
[2] `https://github.com/SafeExamBrowser/seb-mac`

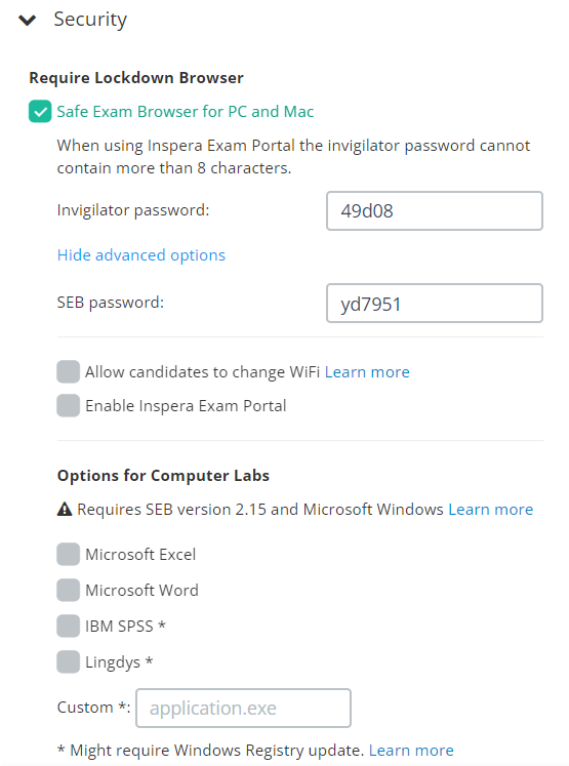

Figure 7.1: Configurable SEB options for an exam in Inspera.

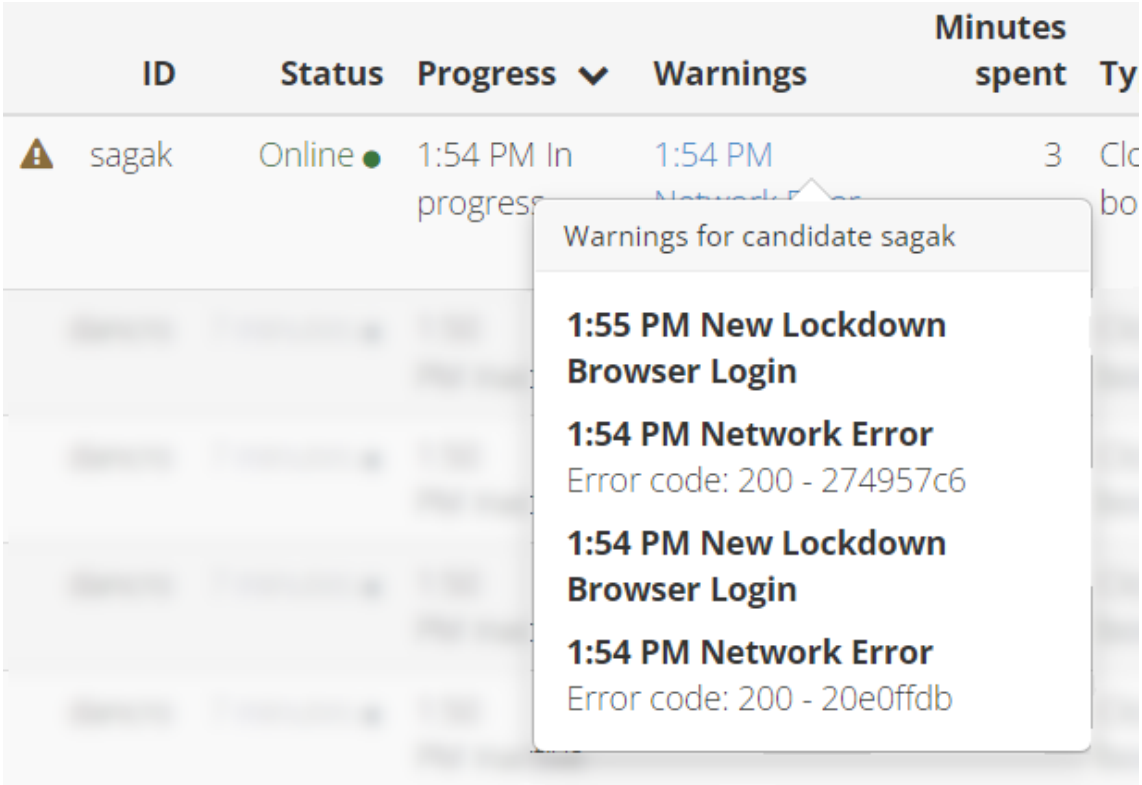

Figure 7.2: Errors generated by an examinee due to suspicious behavior, shown in the monitor tab in the Inspera administrator panel.

7.1.3 Overview

As mentioned earlier in the chapter, the e-exam solution used at Chalmers is a combination of SEB and the Inspera LMS. In Chapter 4, the overall architecture of SEB was explained, along with an architecture diagram shown in Figure 4.1. Here, an overview of the interaction between SEB and Inspera is shown instead in Figure 7.3, along with how the transition from a normal web browser to SEB works.

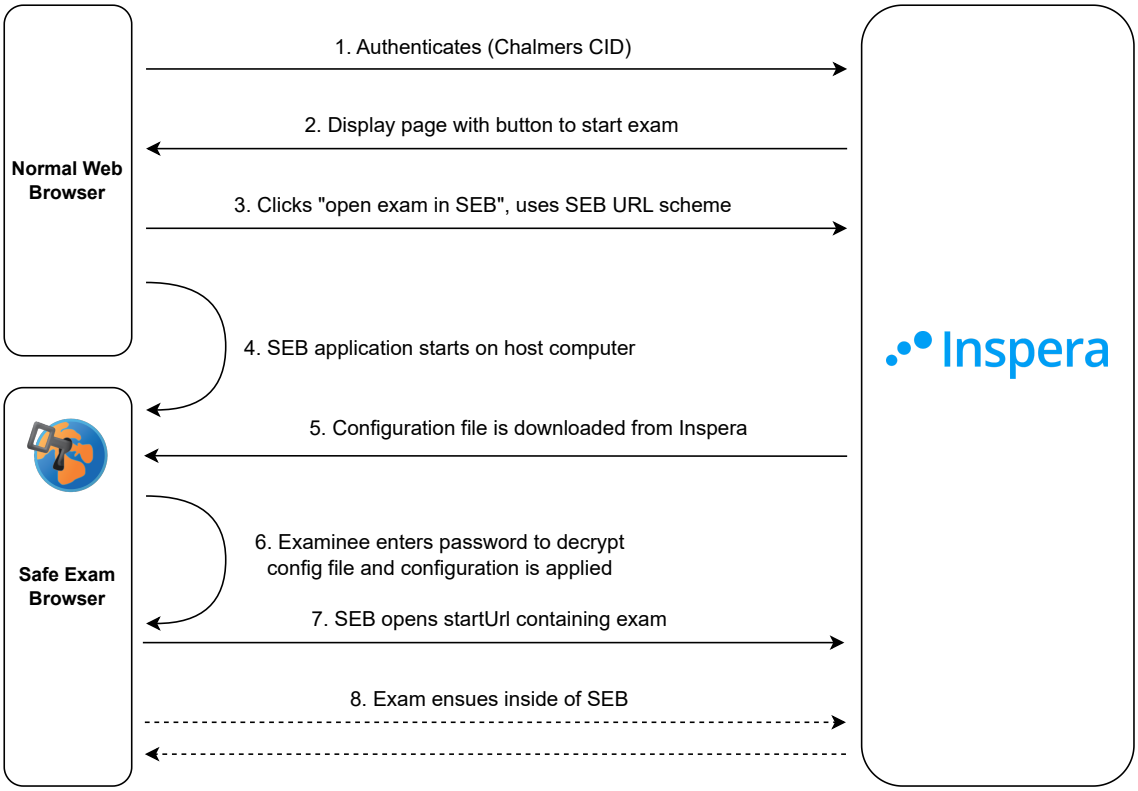

Figure 7.3: Flow between an examinee (left) and Inspera (right) when starting an exam.

As a first step, the examinee will use a normal web browser to navigate to Inspera and authenticate themselves as a Chalmers student through a login page. Once an examinee is authenticated, Inspera will display a page with all previous and upcoming examinations for that examinee. The examinee will then navigate to the exam they are going to write, and is then presented with a button that reads “Start test in Safe Exam Browser”. Upon clicking this button, the browser will navigate to a URL similar to `seb://chalmers.inspera.com/exam-config-file.seb`. This prompts SEB to start on the host machine, which automatically downloads the configuration file at `http://chalmers.inspera.com/exam-config-file.seb`. This URL scheme functionality is also further described in Section 4.3. Then, the user is presented with an input text box where they must enter the password needed to decrypt the configuration file. This password is usually given to examinees at the start of the exam. Finally, SEB uses the `startUrl` in the decrypted configuration file to show

the examinee their exam, as explained in Section 4.2. From this point on, the examinee is locked down inside of SEB writing their exam, until they finally exit SEB after turning in their answers.

It is this interaction between SEB and Inspera that our case study will partially focus on, as it closely resembles the relationship that the CAT showed in Figure 5.7 displays. By investigating the communication between SEB and Inspera, and their interaction with each other, it might be possible to successfully deploy some of the threats from the CAT in Figure 5.7.

7.2 Proxy

In order to be able to investigate the network traffic sent from SEB to Inspera, we built a *proxy server*[3]. The proxy server has been a central component in a few attacks, more specifically, it has allowed us to investigate the network traffic and change the traffic on the go. It works in a *man in the middle* (MITM) manner, with the possibility of eavesdropping and altering the traffic sent between Safe Exam Browser and Inspera. Figure 7.4 shows a graphical overview of what the flow looks like between the corresponding parties. As seen in the figure, the proxy is just forwarding messages between SEB and Inspera, without altering them. However, as mentioned, the proxy is also capable of altering the messages, which will be further discussed in later sections.

Figure 7.4: Flow between Safe Exam Browser and Inspera, where the Proxy Server works in a MITM manner, having access to all traffic sent between the two parties.

We built the proxy using Express.js[4], which is a popular Node.js[5] framework for quickly building web servers. The importance here is not what technologies have been used for building it, we simply chose to use Express.js because both of us had prior experience with it. The same end result could probably have been accomplished using any other programming language or relevant framework.

As explained in Subsection 7.1.3, the URL of the webpage that SEB will show (the `startUrl`) is provided by Inspera, which is under the `https://chalmers.inspera.com` domain. In order for the network traffic originating from SEB to be routed to the proxy instead of Inspera, we had to specifically configure the DNS `hosts` file, which is used during the DNS lookup process whenever your machine is trying to send a

[3] `https://fortinet.com/resources/cyberglossary/proxy-server`
[4] `https://expressjs.com/`
[5] `https://nodejs.org/en/`

web request. This file can be found at `C:/Windows/System32/Drivers/etc/hosts` on Windows, and at `/etc/hosts` on MacOS. All that is required is that the line `127.0.0.1 chalmers.inspera.com` is appended to this file, given that the proxy server is running on `localhost` (127.0.0.1).

Since Inspera uses the HTTPS protocol [58], this means that it is also using TLS/SSL protocol [59]. This implies that our proxy server needs to run HTTPS, while also needing a *SSL certificate* to prove to SEB that it indeed is `chalmers.inspera.com`. The following subsections will further elaborate on how to forge such a certificate and why this works.

7.2.1 Forging SSL certificate for Inspera

Since we have manually configured our traffic to `https://chalmers.inspera.com` to be routed to our proxy instead of the Inspera web server directly, the proxy server needs to *prove* to SEB that it indeed is `chalmers.inspera.com` and not an impostor. In HTTPS, this process is done in the TLS/SSL handshake via *SSL certificates*. A simplified diagram of the TLS/SSL handshake is shown in Figure 7.5.

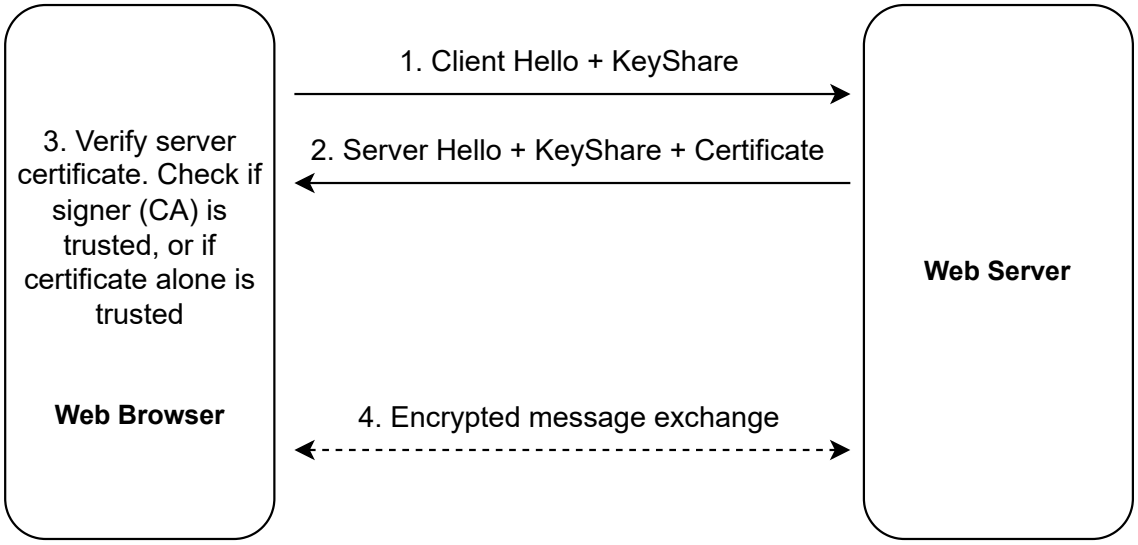

Figure 7.5: A simplified diagram of messages between a web browser and web server during the TLS 1.3 handshake.

There are four steps in the process that needs to be fulfilled in order to forge a certificate for our proxy. First of all, we need to create a *Certificate Authority* (CA) to be able to sign any certificates we create for our proxy. After creating the CA, we now need to create a certificate to be used by our proxy. Once this certificate has been created, the last step is to *sign* this certificate using the newly created CA. After doing all of these steps, we just need to make sure that our proxy uses HTTPS along with our newly created certificate.

In order to both create and sign the certificate(s), we used OpenSSL[6], which is

[6] `https://openssl.org/`

an open-source toolkit used to generate keys, certificates and perform signing. To begin with, we created a new CA called `Cheaters CA`. The naming here is not important, one can simply choose to call it whatever they would like. The next step was to create a certificate for our server (proxy). We looked at the official certificate provided by `chalmers.inspera.com`, and simply copied all alternative names and included them in our certificate. In reality, since SEB only makes requests to `chalmers.inspera.com`, the only required common name would be `chalmers.inspera.com`. However, we chose to include them all simply due to being unsure whether we needed them all or not. The names we included are displayed below:

```
DNS.1 = inspera.no
DNS.2 = *.inspera.se
DNS.3 = *.inspera.com
DNS.4 = *.inspera.no
DNS.5 = *.inspera.de
DNS.6 = inspera.de
DNS.7 = inspera.se
DNS.8 = inspera.com
```

The final step was to sign the newly created certificate using our `Cheaters CA` and turn on HTTPS on our proxy server along with passing it the certificate. However, there is one significant part still missing, one has to specifically *trust* the `Cheaters CA` we created. As shown in Figure 7.5, after a browser retrieves a certificate from a web server, it will now try to verify whether the certificate is legitimate or not. In order for it to verify that the certificate is correct, it needs to trust the CA.

7.2.2 Making SEB trust the Certificate Authority

As mentioned in Chapter 4, SEB for Windows uses the Chromium browser engine. Chromium uses the Chrome Root Store [60], which contains the set of CA certificates that are trusted *by default*. The full list of certificates trusted by the Chrome Root Store can be found in the Chromium docs[7]. In addition to this, Chromium also trusts locally installed root certificates, which means that the way to make Chromium/SEB trust our `Cheaters CA` root certificate, is to trust it locally on our machine.

SEB for MacOS uses WebKit, as explained in Chapter 4. On MacOS, WebKit will use the KeyChain for trusted CA certificates. Since the KeyChain can be modified by a user, our `Cheaters CA` root certificate can also be trusted locally on our MacOS machine.

In earlier versions of SEB for Windows, before Chromium was introduced, XULRunner was used. As investigated by Heintz in 2017 [13], this made it impossible to make SEB trust the CA certificate he created for a proxy he used at that time, meaning that building such a proxy was not possible back then. This leads us to

[7] `https://chromium.googlesource.com/chromium/src/+/main/net/data/ssl/chrome_root_store/root_store.md`

the conclusion that when SEB started using Chromium instead of XULRunner, they actually made the application *less secure*.

7.3 Proxy injection

The first attack we tried using our proxy (see Section 7.2) was to inject information into the SEB environment. This attack falls under the category of spoofing the EEE, which can be seen in the attack tree in Figure 5.2. The same attack was attempted by Heintz [13] in 2017, but was only successful on MacOS at the time, due to the reasons discussed in Subsection 7.2.2.

Injecting information into the SEB environment means that we are now looking at a completely different approach than before: rather than trying to break out of the environment, we try to include prohibited information into the environment. As mentioned before, a vital part of this attack is the proxy we created in Section 7.2. This implies that if the LMS runs over HTTPS, one must be able to forge a certificate for the proxy as we did for Inspera earlier.

The attack works as follows. After setting up our proxy and making sure that SEB was able to communicate with the Inspera server at `chalmers.inspera.com`, we began modifying what the proxy was sending back to SEB. As seen in Figure 7.4, the "unmodified" proxy is just forwarding messages between the two parties *without* altering them. In order for the proxy to inject information into SEB, it needs to *alter* the responses sent from `chalmers.inspera.com`. This means that the proxy is now modified to be active, rather than passive. This flow of information is shown in Figure 7.6.

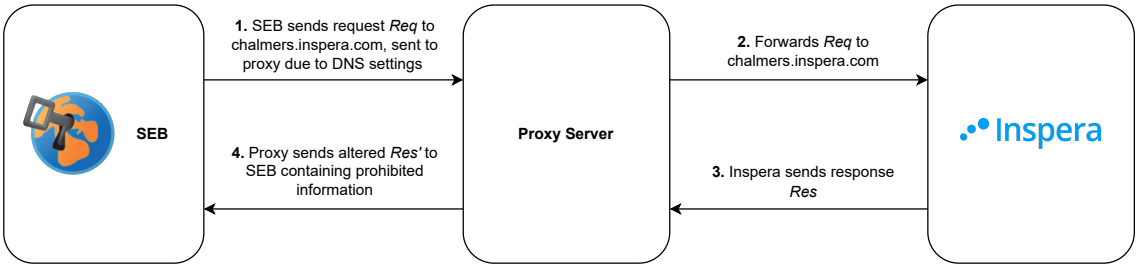

Figure 7.6: Flow of the proxy injection attack.

Figure 7.7 shows an example of such an attack, where we have injected the string "Tiger team was here" into the exam environment, by modifying the response to the `/GetQuestionSet` Inspera API call made at the beginning of the exam. This endpoint is called at the start of every exam when retrieving the questions for the particular exam. This shows that it is indeed possible to inject *any* information into the environment, such as prohibited notes.

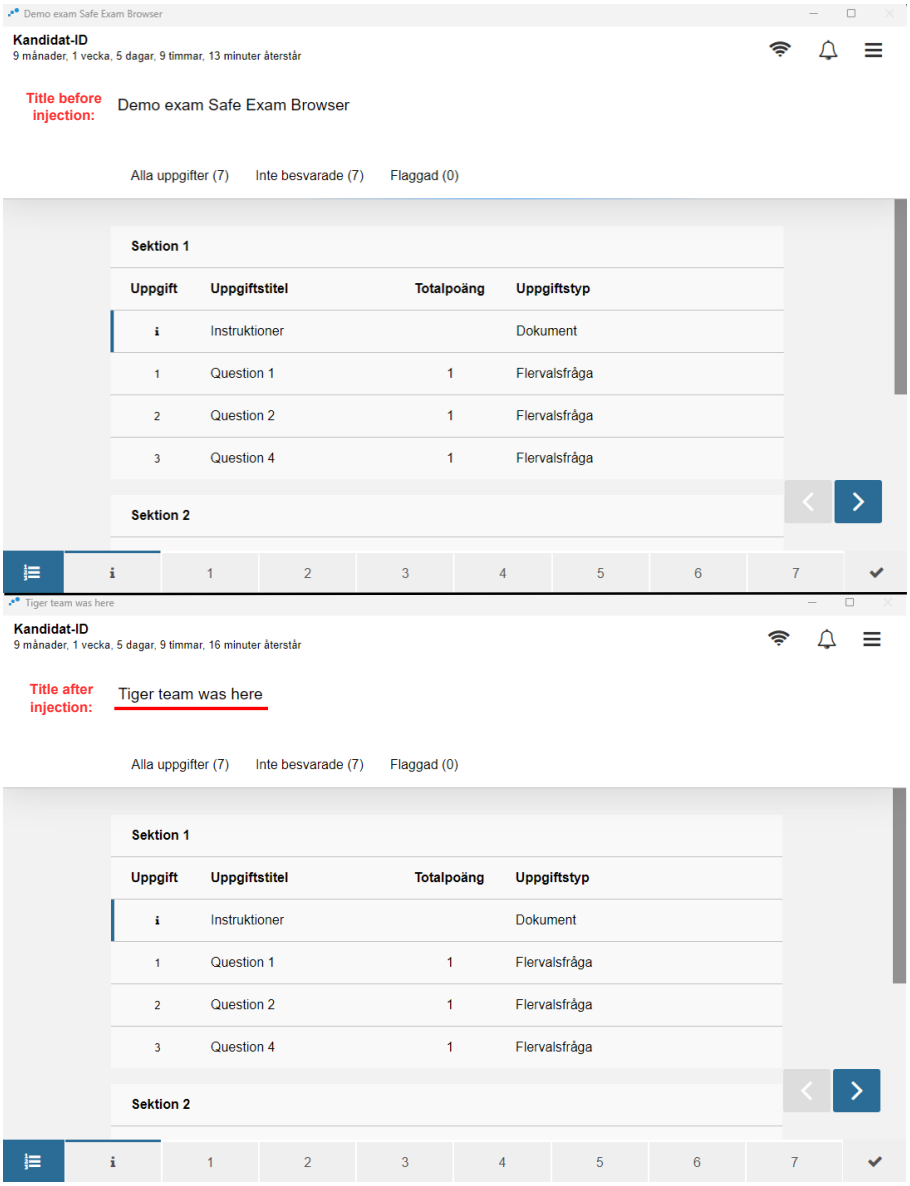

Figure 7.7: Example of injecting the string *"Tiger team was here"* into an Inspera exam. The original exam is shown at the top, whereas the modified exam is shown at the bottom.

One could go as far as injecting a new button into Inspera that looks like it is part of the actual exam. Clicking the button would take you to a new tab of the exam, which in turn would contain several cheating materials that an examinee would need in order to successfully cheat. It is even possible to include new webpages, by using the `<iframe>` HTML tag. Figure 7.8 shows an example of injecting Bing[8] into the environment. However, all webpages cannot be rendered in an `<iframe>`, due to the `X-Frame-Options`[9] header that is widely used today, which is used to *prevent* a webpage from being able to be rendered inside of an `<iframe>`.

[8] `https://bing.com/`
[9] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options/`

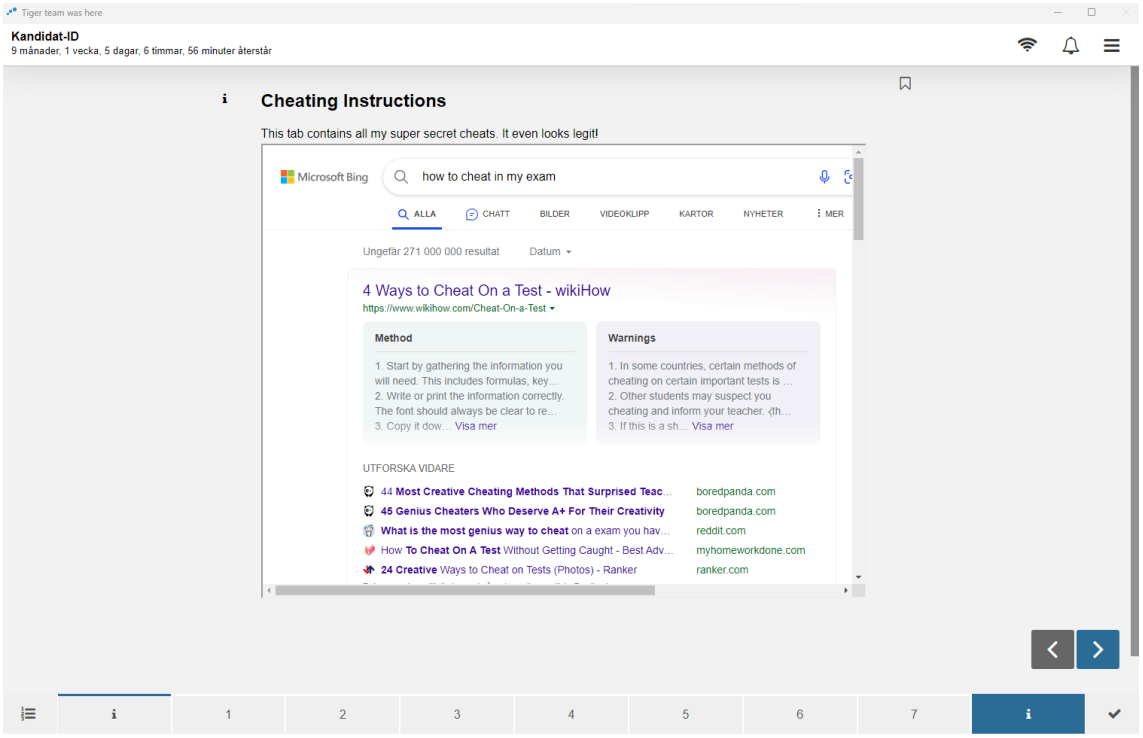

Figure 7.8: Example of injecting Bing into a new exam tab that freely allows an examinee to search the web.

7.3.1 Ignoring local certificates and using exam networks

The underlying problem of why this attack is possible in the first place is the proxy. Without the proxy, it would be impossible to inject information using the approach described above. As seen from the attacks presented in this section, there exist various approaches for an examinee to cheat through the use of the proxy. Therefore, in order to stop all of these attacks, an examinee must not be able to use such a proxy.

As presented in Section 7.2, in order for the proxy to work, SEB needs to *trust* the CA that has signed the proxy server's certificate. This is easy to spoof today since one can just trust the CA locally on their computer, which makes Chromium trust the CA. For this certificate spoofing to not work, SEB would need to use a browser engine that does not support locally trusted root certificates. This would allow SEB to notice when an examinee receives an untrusted certificate from a server, hopefully raising some alarms of potential cheating behavior.

Stopping an examinee from injecting new webpages in an `<iframe>` could be done by running the in-hall exam on a dedicated exam network. Such a network could be set up so that *only* webpages relevant to the exam could be accessed. This would require all examinees to be connected to the exam network, and would also require additional security access checks on Inspera to check whether the examinee is connected to the correct network. However, this injection only works if the proxy is a viable attack, and if the examinee is unable to use a proxy to begin with, then

this mitigation is redundant.

This attack and mitigation suggestion regarding local certificate trust has been reported to the developers of SEB[10], with an acknowledgment of the severity of this attack. However, the developers argue that such a mitigation implementation is not realistic to implement in the current state of SEB, leaving it for the future.

7.4 Brute-forcing the config file

Like explained in Section 4.2, the configuration file is encrypted using the RNCryptor framework[11]. RNCryptor allows the use of passwords for encryption by applying PBKDF2 [35], a password-based key derivation function. This attack aims to find the password used to encrypt the configuration file, in a brute-force manner. If an examinee is able to decrypt the configuration file ahead of time, they will have more time to prepare other attacks that require the configuration file, such as some of the attacks described in Section 7.5.

Napier, the author of RNCryptor, goes into great detail about the complexity of brute-forcing passwords in his framework [61]. Napier reveals that the majority of the time during encryption and decryption is spent deriving the key from the password, with PBKDF2. RNCryptor uses 10000 rounds of PBKDF2 to derive a key from the password, which takes around 10 milliseconds on one core of Napier's machine of choice. Napier, therefore, concludes that a brute-force search could test around 100 passwords per second per core on his machine.

With this reasoning, it is trivial to calculate how long a password should be to withstand a certain threshold of time before being brute-forced. We will define 1 *core-year* as the computational effort that a single CPU core run for 24 hours of the day for 365 days can perform. 1 core-year is also equivalent to 365 CPU cores being run for 24 hours, or 8760 cores running for 1 hour.

Given a set $|S| = 62$ of all lower and uppercase alphanumeric characters to use in our passwords, and that it should take 100000 core-years for it to be brute-forced, the required length L of a password can be calculated as follows.

We first realize that the amount of passwords that a core can test per year is $100 \cdot 60 \cdot 60 \cdot 24 \cdot 365 = 31536 \cdot 10^5$. Our constraint is 100000 core-years, meaning that it must take 100000 years for a single core to test every possible password, so the amount of possible passwords needs to be $31536 \cdot 10^{10}$. With this, we must only solve the trivial equation $|S|^L = 62^L = 31536 \cdot 10^{10}$, giving us $L = 8,0891$. However, we must remember that the resulting L must be integral, so we round up to the nearest number $L = 9$. With this, we found that the password must be at least 9 characters long for it to require 100000 core years of brute-force.

[10] `https://github.com/SafeExamBrowser/seb-win-refactoring/issues/598`
[11] `https://github.com/RNCryptor/RNCryptor/`

7.4.1 Inspera passwords

The configuration files that Inspera creates for examinees are encrypted with a password selected by Inspera. We were able to conclude that the password generated by Inspera is always two lowercase letters followed by four digits. This can be seen in Table 7.1, which shows the generated passwords for 10 different exams. This gave us the confidence that the structure of these passwords is always the same, giving us a very specific key space to work with for our brute-force search.

Exam	Password
1	`iw0510`
2	`ie2878`
3	`qv8996`
4	`ra4685`
5	`xh5200`
6	`ko7052`
7	`df1925`
8	`ge3582`
9	`yd7951`
10	`nm5846`

Table 7.1: List of passwords generated by Inspera which are used to encrypt SEB configuration files.

Finally, we were also told by Inspera administrators at Chalmers that they will almost always use the generated password generated by Inspera. However, there may be occasions where an examiner or administrator manually changes the password, but the administration told us that this happens very rarely. Therefore, this attack only works for configuration files where the generated password has not been changed.

7.4.2 Brute-force attack

The attack described in this section corresponds to the *brute-force encryption key* attack in the CAT in Figure 5.5. It is important to note that this attack is not made possible due to a weak cryptographic implementation, it is made possible due to improper usage of the cryptographic implementation.

The investigation done in Subsection 7.4.1 showed that the passwords always follow a certain structure of two lowercase letters followed by four digits. This gives us a password space size of $6760000 = 26^2 \cdot 10^4$. Generating these passwords is trivial, but testing them is not as trivial.

We were able to use the same C# code that SEB uses to perform its encryption and decryption of configuration files, to perform this attack[12]. This allowed us to very quickly get a working brute-force search to work, rather than needing to

[12] `https://github.com/SafeExamBrowser/seb-win-refactoring/blob/b69280731a212cad699f911f98431c5d47104d7b/SebWindowsConfig/Utilities/SEBProtectionController.cs`

write our own code that uses the RNCryptor framework. A single password test in our brute-force search consists of selecting a non-tested password, decrypting the configuration file data with the selected password, and checking if the first three bytes of the decrypted data are `0x3C 0x3F 0x78`. The first three bytes being equal to that sequence means that the bytes encoded as UTF-8[13] is the string `<?x`. This allows us to quickly check if the data is valid XML, revealing if the password is valid or not.

During testing, we measured that our brute-force search was able to test a single password in 20 milliseconds on one core. This meant that we would be able to test 50 passwords per second per core on the machine it was run on. With access to 20 cores on the machine, it was possible to test 1000 passwords per second.

Given the password space size of 6760000 and a testing speed of 1000 passwords per second, it is possible to brute-force the configuration file created by Inspera after 6760 seconds, which is roughly two hours. During testing, an example configuration file was brute-forced after a little over one hour, which is around the expected time. According to the Inspera administration at Chalmers, the configuration file for the exam is generated and made available 2-3 days before the actual exam, making this attack more than feasible.

7.4.3 Improving password structure

To mitigate this attack, we will calculate the password space size that is required for an appropriately chosen brute-force effort, and then recommend a new password structure that results in at least the calculated password space size. This mitigation is completely up to the Chalmers exam administration to implement, as SEB or RNCryptor is unable to implement this. However, it might even be of interest for Chalmers to report to the developers of Inspera that there should be different options for password generation to choose from. That way, different institutions will be able to use different types of passwords depending on their requirements.

We will assume that an examinee is able to utilize 20 cores for their brute-force search and that it is possible that several examinees may collaborate in their brute-force search. If we also assume an upper bound of 300 examinees for a given exam (a pretty large course at the CSE department at Chalmers has approximately 300 students), that gives the collaborating examinees a total of 6000 cores for their brute-force search. This means that during a three-day period, the examinees can exert an 18000 core-days brute-force effort.

If an examinee is able to test 100 passwords per second per core ($100 \cdot 60 \cdot 60 \cdot 24 = 8640000$ passwords per day per core), then 18000 core-days of brute-force will be able to test $15552 \cdot 10^7$ passwords. If we decide our password character set to be $|S| = 62$, all lower and upper case alphanumeric characters and digits, then we would need passwords of length 7 to withstand 18000 core-days. In fact, a password of length 7 can even withstand > 400000 core-days of effort. Passwords of length 7

[13] `https://developer.mozilla.org/en-US/docs/Glossary/UTF-8/`

from the password character set S makes it *infeasible* for examinees to brute-force a configuration file.

7.4.4 Revealing the configuration file

Finally, a different way to mitigate this attack is to make the configuration file available at the same time as the password is given to the examinees, rather than 2-3 days before an exam. However, if the configuration file is revealed at the same time as the password, no encryption would be needed, since an examinee is supposed to decrypt it using the password at that point anyway. Therefore, it could be possible to distribute unencrypted configuration files at the start of the exam, with a different password mechanism. Such a password mechanism could be to store the hash of a password inside of the configuration file, which can be compared with an entered password to grant an examinee access to the exam. The only thing encryption adds is the possibility to distribute the configuration file before the exam.

7.5 Accessing Inspera exam outside of SEB

SEB could be circumvented completely if an examinee could access an exam *without* even using it. This would also imply that they would have access to any prohibited material throughout the exam since they are taking the exam outside of the locked environment. This is one of the more important aspects of SEB in terms of security – it does not matter how secure the actual locked environment is if one is able to choose to never use it in the first place.

This investigation has revealed two *critical* vulnerabilities, both in how Inspera validates that an examinee is indeed using SEB, but also what "tools" SEB provides to authenticate SEB usage.

7.5.1 StartUrl vulnerability

When investigating the decryption of the SEB config files provided by Inspera, discussed in Section 7.4, we discovered a critical vulnerability in *how* Inspera validates that an examinee is using SEB. As mentioned in Section 4.4, the LMS is responsible for using all the tools that SEB provides in order to authenticate that SEB is being used. The SEB documentation even includes a comprehensive guide of how to do so [57]. This shows that SEB is only able to provide the relevant tools for an LMS to validate the correct SEB usage, but if an LMS actively chooses to *not* validate anything, this leads to vulnerabilities.

After decrypting a config file obtained from Inspera, we noticed that the `startUrl` contains a direct log-on link in order to authenticate the examinee to Inspera.

```
https://chalmers.inspera.com:443/login?action=authenticate
&userName=<chalmerscid>&directLogonKey=<keyhere>&marketPlaceId=<id>
&requestedURL=<examUrl>
```

Upon visiting this link, the examinee will immediately be authenticated to Inspera and will be redirected to the `examUrl` parameter provided in the link. This is the exact same procedure that happens when an examinee opens SEB using a config file containing this `startUrl`. When visiting this link in a regular browser, a page is shown containing a `Start test` button. The page is shown in Figure 7.9. Upon clicking the `Start test` button, we are redirected to `https://chalmers.inspera.com/player/?assessmentRunId=<examId>&context=exam`. This is the URL that tries to start the exam, but since we are not using SEB, we currently see "Your browser does not pass the requirements". We can also see this in the response of the `/StartTest` API call, which contains the two parameters `userHasSeb:false`, and `passesLockDownRequirements:false`.

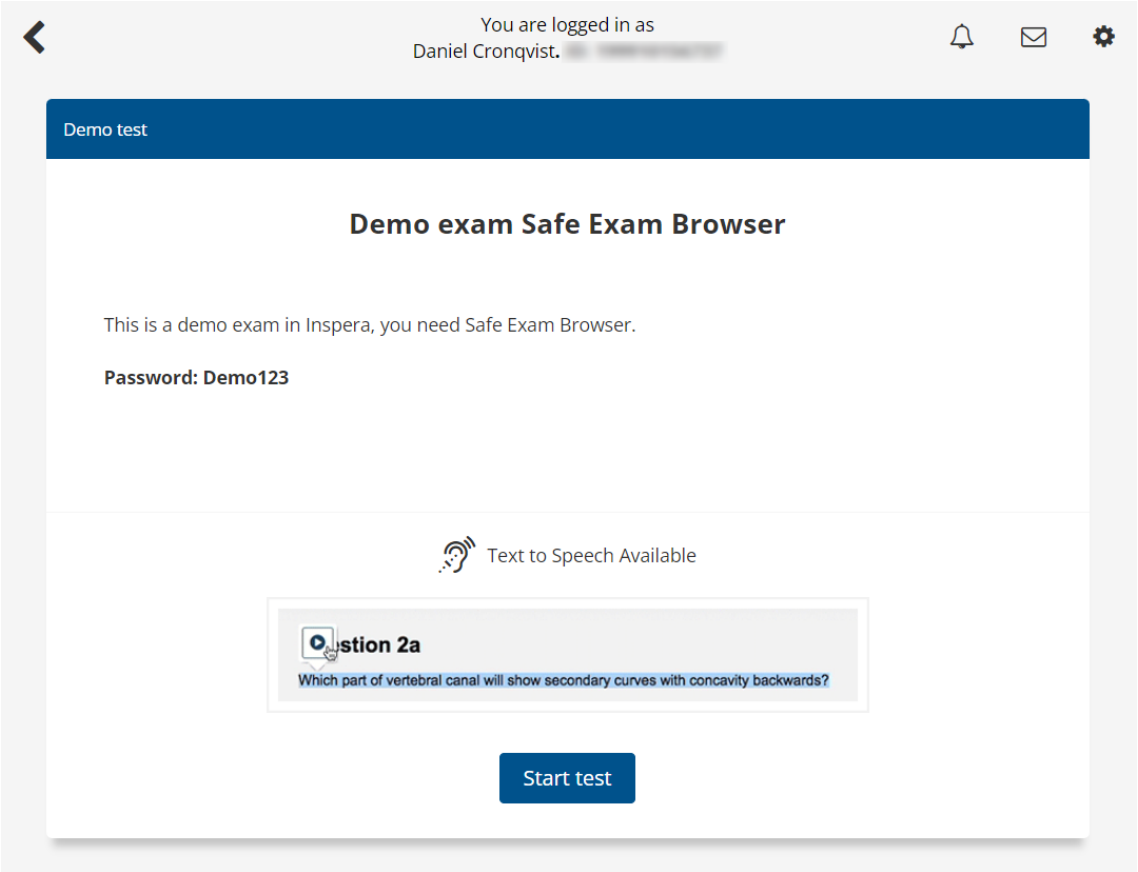

Figure 7.9: Visiting the `startUrl` for the demo exam in Google Chrome.

This made us believe that we need to add the same security parameters to our requests as SEB does. As mentioned in Section 4.5, there exists three headers, `User-Agent`, `x-safeexambrowser-configkeyhash`, and `x-safeexambrowser-requesthash`. Starting off, we appended the `User-Agent` header shown in Table 4.2 to our requests. This was done by using Burp Suite[14], but could be done by using any proxy server, such as the one we constructed in Section 7.2. Initially, we did not expect *only* this header to be enough, since we assumed that Inspera *most*

[14] `https://portswigger.net/burp/`

likely validated the other two headers as well since it would be very easy to bypass otherwise. However, we were completely wrong, and it turns out that all that is required is the right `User-Agent` header. Upon visiting the `startUrl` again, Inspera now believes that we are using SEB, as seen in Figure 7.10. Pressing the `Start test` button starts the exam as usual and the response of the `/StartTest` API call now instead contains `userHasSeb:true`, and `passesLockDownRequirements:true`.

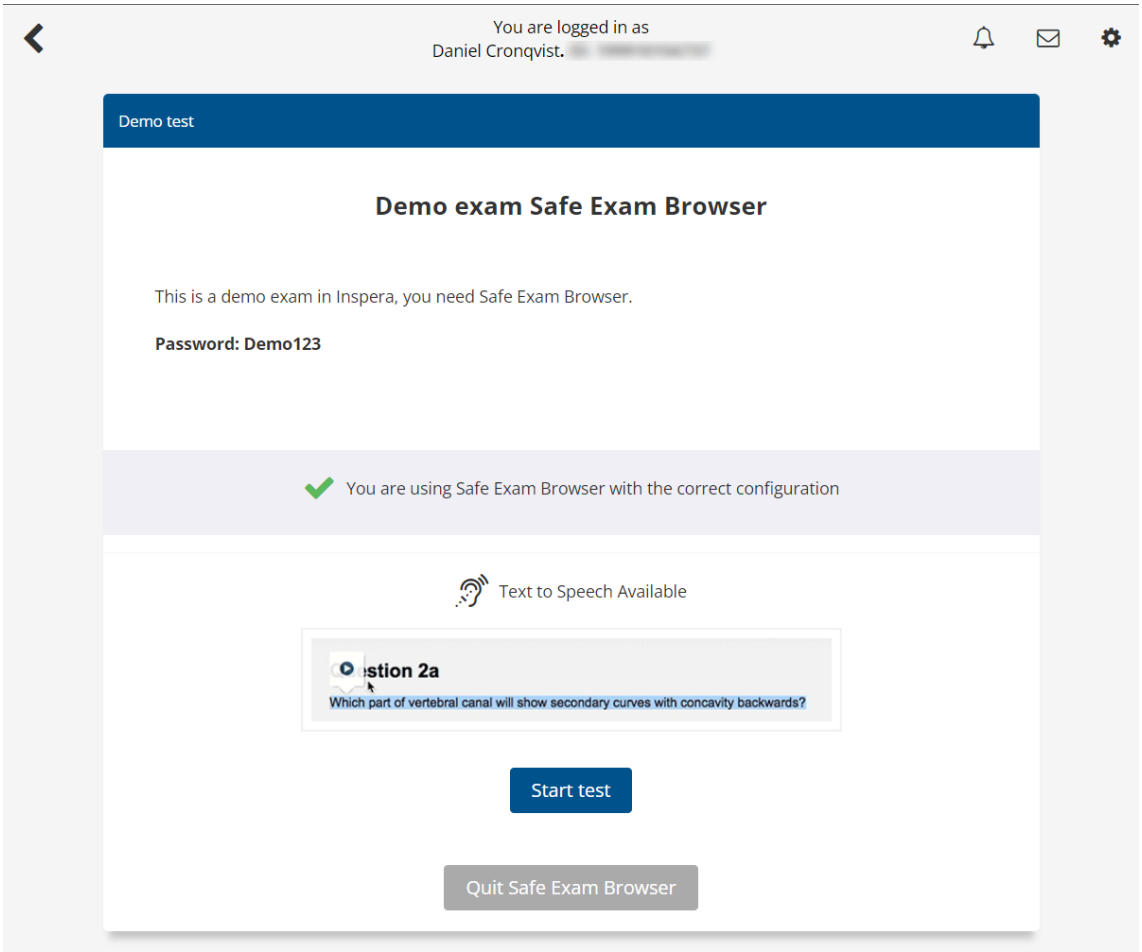

Figure 7.10: Visiting the `startUrl` for the demo exam in Google Chrome, after appending the `User-Agent` header to our requests.

This vulnerability is *extremely* bad, and it is also very easy for an examinee to pull off. This leads us to the conclusion that currently it is *trivial* for an examinee to access an exam outside of SEB. From the Inspera point of view, the examiner has no way to determine whether an examinee is actually using SEB or if they have just appended the correct `User-Agent` header to their requests. The monitor tab on Inspera shows that the examinee is currently using SEB regardless of the approach. The only way to spot that an examinee is cheating using this approach is if an invigilator would catch them doing it in the exam hall. From our perspective, this can be quite hard for an invigilator to catch, since one could just open up the exam in full-screen mode in a regular browser, and access the cheating-related materials whenever the invigilator is busy doing something else.

Inspera should make use of all tools that SEB facilities for authenticating correct SEB usage. In its current state, it is unacceptable that Inspera completely ignores the headers that SEB provides, except for the `User-Agent`. However, Subsection 7.5.2 will show that an examinee is able to re-construct the two other headers trivially, which ultimately leads us to point out the fundamental flaw in SEB.

7.5.2 Re-constructing the SEB security headers

Even though we managed to access the exam outside of SEB simply by appending the correct `User-Agent` header to our requests, we were wondering if we could access the exam given that Inspera would use all three headers shown in Table 4.2. This would require us to re-construct the exact same values of the two `x-safeexambrowser-configkeyhash` and `x-safeexambrowser-requesthash` headers, and then append these headers to our requests.

Recall the explanation from Section 4.5 of how these two headers are calculated. In order for us to be able to reconstruct the exact same values, we first needed to calculate the exact same `Browser Exam Key` and `Config Key` that is used for a specific config file and SEB version.

As previously explained in Section 4.5, the `Config Key` is calculated by taking the SHA256 hash of the `SEB-JSON` of a decrypted config file. In order to retrieve the decrypted config file, one can either crack it by brute-force, as seen in Section 7.4, or simply decrypt it when receiving the exam password immediately before the exam starts. Similarly, the `Browser Exam Key` is equally as easy to obtain. Since the key depends on the actual SEB version that is used, the examinee only needs to download the correct SEB version, and then use the code from the SEB codebase shown in Figure 4.4 to calculate the key. Therefore, it is trivial to reconstruct both of the two headers, meaning that it is trivial to access an exam outside of SEB even if the LMS would validate these headers.

We used our proxy constructed in Section 7.2 to calculate these headers and append them to our requests. Since Inspera only validates the `User-Agent` header, we were unable to test this in "real-time". Instead, we opened up a specific exam in SEB and logged all the headers sent to the corresponding exam URLs. After this, we re-constructed the two headers using our newly calculated `Browser Exam Key` and `Config Key` and were able to determine that they had the exact same values for all the different URLs.

7.5.3 The flaws of the SEB design

As shown in the previous section, Subsection 7.5.2, the current security measures provided by SEB for an LMS to authenticate correct SEB usage are not sufficient. We have shown that it is trivial to reconstruct the current headers provided by SEB, which implies that SEB in its current state is very easy to bypass. The only way of proving to an LMS that an examinee is using SEB is via the headers, and if an examinee is trivially able to re-construct them it implies that the SEB software contains major flaws.

These flaws stem from the fact that an examinee is given the decryption password at the start of the exam. They could simply prepare everything that is needed before entering the exam hall, decrypt the config file upon receiving the password, and then access the exam through a regular browser by re-constructing the headers and appending them to the corresponding HTTP requests. One could compare this to two parties communicating via encrypted messages, using a symmetric encryption scheme, where the key is made public but the data is assumed to be secure. It is almost as if SEB assumes that an examinee never uses the password in a malicious manner. The security assumption of SEB therefore currently lies in the fact that an examinee will not act maliciously, which completely contradicts the use of SEB, since an institution likely uses SEB as examinees cannot be trusted.

The flaw has previously been pointed out by Heintz [13] in 2017, however, SEB only facilitated the `x-safeexambrowser-requesthash` calculated by the `Browser Exam Key` at the time. It seems like the developers of SEB have made an effort to harden the security of SEB by introducing additional headers in later versions of SEB. Our opinion is that it does not matter *how many* headers you introduce if they all rely on the same decryption key, which is given to the examinee in the end. Furthermore, Heintz concludes the problem with "No matter how well you hide the secret, at some point in time the secret needs to be available for Safe Exam Browser to send to Inspera Assessment, and as such at that point, it will be available to the computer it is installed". From our point of view, this quote makes it sound like there is no solution to this major SEB flaw. However, we disagree, as we have found new mitigation methods which will be presented in Chapter 8.

7.6 Modifying Windows registry for process lookup

This attack specifically targets the third-party application part of SEB on Windows. The same attack cannot be executed on MacOS, since SEB for MacOS lacks the corresponding functionality for third-party applications as mentioned in Subsection 4.2.1. As discussed in Subsection 4.2.1, SEB can whitelist certain applications to be launched during an exam, by displaying clickable buttons in a taskbar-like interface. The attack specifically tries to disguise malicious applications as whitelisted ones, allowing an examinee to open other applications than the ones whitelisted.

7.6.1 SEB process lookup

Currently, SEB performs polling on a regular interval to retrieve all running processes on the host computer with the help of the `System.Diagnostics` namespace in C#. An excerpt of this polling can be seen in the code below, taken from GitHub[15]. By comparing the retrieved processes between polling intervals, SEB can determine which processes that have been started, and which have been terminated. For all new processes that have been started, it performs a check to see whether the process

[15] `https://github.com/SafeExamBrowser/seb-win-refactoring/blob/cba73bd7272c5b85e15952876f62139153cc7ae7/SafeExamBrowser.Monitoring/Applications/ApplicationMonitor.cs`

is a blacklisted or whitelisted application. Blacklisted applications are terminated immediately, whereas whitelisted applications are allowed to run. The way that SEB performs these checks opens up a possible attack vector in the application: it *only* looks at the name of the executable file that was used to start the process. This means that the name of the executable is the only kind of authentication of the process. The authentication is relatively easy to bypass since an examinee could simply rename any executable to the whitelisted name, and then SEB would think that it is whitelisted.

```
// Excerpt from SEB source code on GitHub
public class ApplicationMonitor
{
    private Timer timer;

    public ApplicationMonitor(int interval_ms, ...)
    {
        this.timer = new Timer(interval_ms);
    }

    public void Start()
    {
        this.timer += Timer_Elapsed;
        this.timer.Start();
    }

    private void Timer_Elapsed(object sender, ElapsedEventArgs e)
    {
        var running = processFactory.GetAllRunning();
    }
}
```

However, it is not as easy as that, since further steps need to be taken to make SEB launch the renamed executable. When registering an application/executable as whitelisted in SEB, you only supply the name of the executable file, for example, `excel.exe`. This means that SEB or the operating system needs to provide some way of mapping this `excel.exe` to an actual executable file on the host computer. SEB uses the Windows Registry for this. The registry key `HKEY_LOCAL_MACHINE\SOFTWARE-\Microsoft\Windows\CurrentVersion\App Paths` contains entries for most, but not all, installed applications on the host. These entries contain the absolute path to the executable file from a given name of an executable. SEB attempts to find an entry matching the whitelisted executable using the method in the below code snippet[16], and if it finds it, it creates a clickable button in a taskbar-like interface at the bottom of the screen. Clicking this button will launch a new process from the executable file that the registry entry points to.

[16] `https://github.com/SafeExamBrowser/seb-win-refactoring/blob/cba73bd7272c5b85e15952876f62139153cc7ae7/SafeExamBrowser.Applications/ApplicationFactory.cs`

```
// Excerpt from SEB source code on GitHub
private string QueryPathFromRegistry(WhitelistApplication settings)
{
    try
    {
        var path = $"SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\{settings.ExecutableName}";
        using (var key = Registry.LocalMachine.OpenSubKey(path))
        {
            if (key != null)
            {
                return key.GetValue("Path") as string;
            }
        }
    }
    catch (Exception e)
    {
        logger.Error($"Failed to query path in registry for '{settings.ExecutableName}'!", e);
    }

    return default(string);
}
```

This means that SEB blindly trusts that the registry points to the correct whitelisted executable, and not some other executable. This can be exploited, since an examinee can modify their Windows Registry entries, and point them to whatever executable they want to, as long as the executable file has been renamed to the whitelisted name as well.

7.6.2 Allowing arbitrary process to be run in SEB

We were able to successfully perform this attack by doing the following.

1. (Prerequisite) Identify at least one whitelisted application in the configuration, e.g. `excel.exe`. This can be done by either brute-forcing the config file as seen in Section 7.4, or simply asking the examiner about permitted applications.

2. Locate an executable that should be launched inside of SEB instead of `excel.exe`, e.g. `C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe`.

3. Copy `powershell.exe` and rename the copy to `excel.exe`, e.g. so that `C:\Windows\System32\WindowsPowerShell\v1.0\excel.exe` exists.

4. Modify `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\excel.exe` in the registry to point to `C:\Windows\System32\WindowsPowerShell\v1.0\excel.exe` . Also add a `Path` key with the value `C:\Windows\System32\WindowsPowerShell\v1.0`. The registry should look like the image shown in Figure 7.11.

5. Now when SEB is launched with a configuration that allows `excel.exe`, an icon for `PowerShell` is created, allowing it to be launched inside of SEB, shown in Figure 7.12.

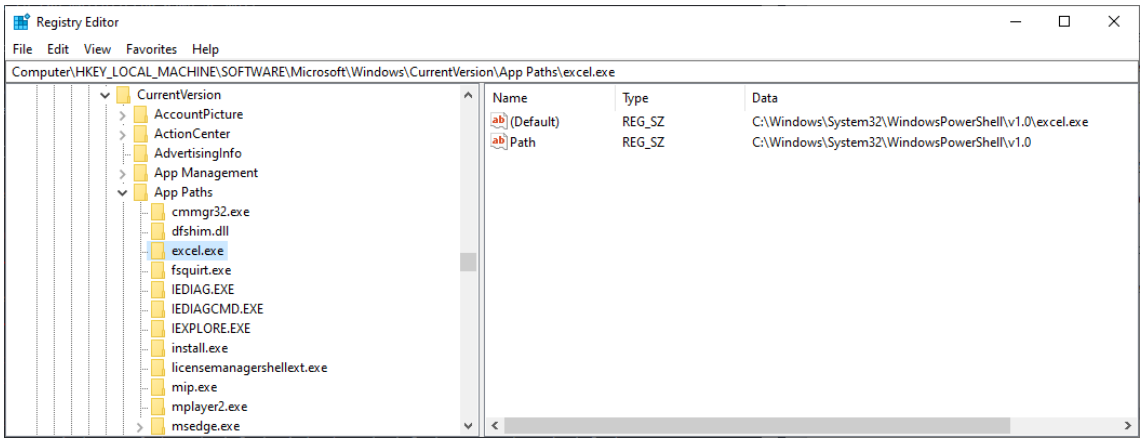

Figure 7.11: Example of modified registry entry for `excel.exe` to launch `PowerShell` instead.

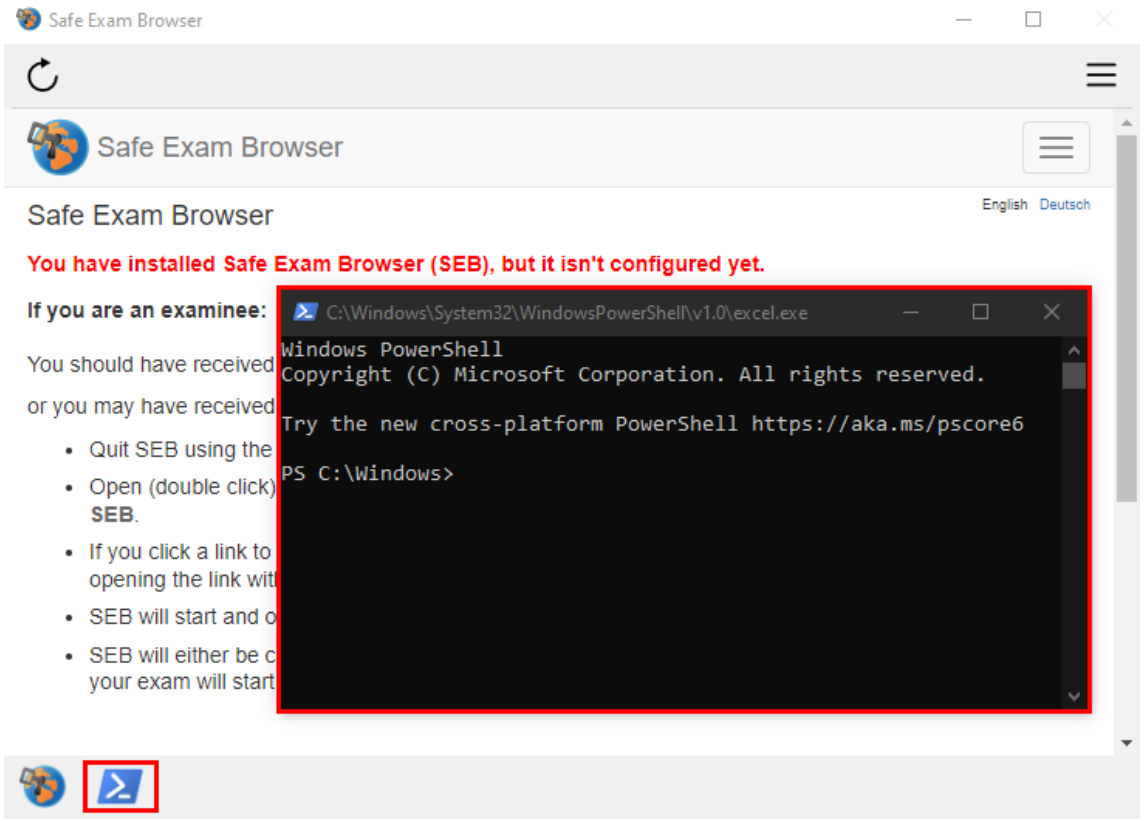

Figure 7.12: `PowerShell` launched inside of SEB after modifying registry.

7.6.3 Launching multiple processes in SEB

Further investigation was done on this attack to see if it could be extended. Odds are that if a specific executable is whitelisted during an exam, it is useful or even necessary to have it during the exam, with or without cheating abilities. Therefore it might be useful to extend this attack to launch multiple processes, where the

correct whitelisted process is one of them. This was found to be possible by using some very simple PowerShell[17] scripting. A PowerShell script can easily launch an arbitrary amount of processes and then be turned into an executable through a tool like PS2EXE[18]. An example of such a script can be seen below:

```
1 Start-Process -FilePath "/path/to/excel.exe" # Whitelisted exe
2 Start-Process -FilePath "/path/to/cheat/excel.exe" # Cheating exe
```

The above PowerShell script could be turned into an executable with the help of PS2EXE, and be named `excel.exe`. Finally, the Windows Registry would need to be modified so that the entry for `excel.exe` points to this executable. Once that is done, SEB will allow you to run the above script, launch two executables that are both named `excel.exe`, and give you both the whitelisted process necessary for the exam, and the cheating enabling process. It is important that every single process that is started comes from an executable that has the same name as the whitelisted executable since it otherwise is likely to be immediately terminated by SEB. As long as there is at least one whitelisted executable for the exam, this attack is possible. It is also relatively easy to prepare since most of it can be done by simple scripting. Such preparation scripts could also be distributed beforehand to other examinees to also let them cheat.

7.6.4 Adding executable signatures

Finally, to mitigate this attack, additional authentication of the executable that is launched by SEB needs to be implemented. If whitelisting an executable required the configuration to include a signature or hash of the executable, the executable file could be verified before launching the process. This would instead mean that the security assumption would lie on the difficulty of forging a valid signature or hash for an arbitrary executable instead. If a strong hash function such as SHA256 was used for this, such a forgery would be infeasible to construct, thus mitigating this attack. It is however important to remember that an attack might still be possible if an attacker is able to successfully tamper with the configuration file to replace the signature, or SEB itself such that the verification always passes. But, if those attacks are possible, then the examinee would likely benefit more from other attacks that yield greater cheating abilities, like disabling lock-down mode, or using SEB inside of a virtual machine.

This attack and mitigation suggestion has been reported to the developers of SEB[19], with a planned implementation of the suggestion as part of a future version of SEB (version 3.6.0).

[17] `https://learn.microsoft.com/en-us/powershell/`
[18] `https://github.com/MScholtes/PS2EXE`
[19] `https://github.com/SafeExamBrowser/seb-win-refactoring/issues/593`

7.7 Injecting text via USB device

Currently, SEB does not check how many keyboards are connected to the computer. This opens up a potential attack vector for an examinee: a malicious examinee may inject text into SEB via a keystroke injection using a USB device that has been reprogrammed to act as a keyboard, also referred to as a *badusb*. This would allow the examinee to inject text consisting of notes written before the exam, similar to bringing a physical cheat sheet to the exam hall. The attack is displayed in the tampering attack tree found in Figure 5.3.

We tested this out using a Flipper Zero device[20], by using its badusb functionality. Like expected, the attack worked, and we were able to inject any information into a text box of the Inspera exam and therefore be able to access a digital cheat sheet inside of SEB.

The attack would require an examinee to bring a USB device into the exam hall in addition to their own computer. This means that they may be spotted by an invigilator when inserting the USB into the computer, but since a USB device is quite small it would also be very possible for an examinee to do this unnoticed.

7.7.1 Adding keyboard count check

In order to prevent this attack, SEB would need to add checks that prevent the software from starting if the examinee has more than one keyboard connected. In addition to the startup check, SEB would also continuously need to poll and check if the user ever connects a new keyboard during runtime. A similar feature is already implemented in SEB, allowing one to define how many screens an examinee may have connected, which is configured in the configuration file.

This attack and mitigation suggestion has been reported to the developers of SEB[21], where they showed interest in implementing such a mitigation in a future version of SEB.

7.8 SEB version enforced by Inspera

As mentioned in Subsection 7.1.1, we have studied Windows version 3.4.1 and MacOS version 2.3.2 of SEB. An examinee is required to use either of these two versions: if you are using the wrong version, Inspera will refuse to startup the exam in SEB. Inspera currently checks what version an examinee is using by examining the version number in the `User-Agent` header.

At the point of writing this thesis, version 3.4.1 for Windows is the newest Windows version of SEB. However, version 2.3.2 for MacOS was released over two years ago, in March 2021. The fact that Inspera still uses a version of SEB that was released over two years ago is very bad. This implies that *any* SEB vulnerabilities that have

[20] `https://flipperzero.one/`

[21] `https://github.com/SafeExamBrowser/seb-win-refactoring/issues/596`

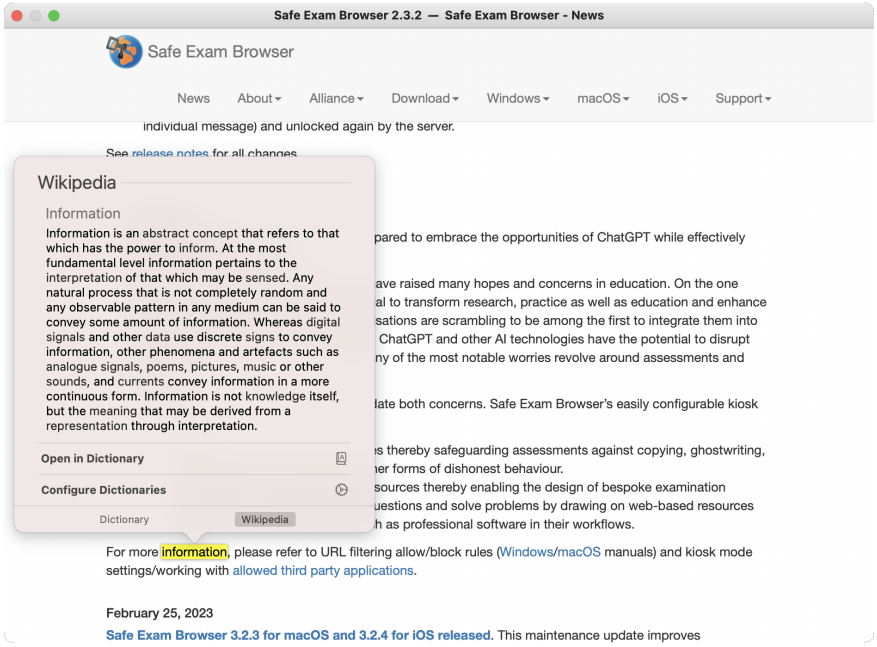

Figure 7.13: Force Touch Dictionary feature on MacOS inside of SEB version 2.3.2.

been patched since then in newer versions still exist in the current version used by Inspera. Therefore, Inspera should consider upgrading to newer SEB versions faster whenever they are released.

7.8.1 Dictionary lookup on MacOS

An issue regarding the ability to perform dictionary lookups of selected words is present in the version of SEB enforced by Inspera, which is claimed to be fixed in a newer version (3.2.3) [62]. By simply selecting text on the displayed web page inside SEB, one can use MacOS's *force touch dictionary* feature to display more information about the selected text. The force touch dictionary feature is able to present you with entire answers to simpler questions, just like performing a simple search in a search engine. It is also able to present you with excerpts from websites like Wikipedia. An example of it being used inside SEB can be seen in Figure 7.13.

7.9 Summary of attack results

Table 7.2 shows an overview of all the attempted attacks in this case study. It is evident that the e-exam solution used at Chalmers suffers from several vulnerabilities that would allow an examinee to cheat.

7.10 Design principles conformity

As shown in Table 7.2, all performed attacks have a corresponding threat identifier from Chapter 5. With this, it is possible to check which principles SEB does not

Attack	Windows result	Mac result	Threat ID(s)
Proxy Injection	Exploited	Exploited	ET11
Brute-forcing the config file	Exploited	Exploited	EI4, EI5, EI1
Re-constructing SEB security headers	Exploited	Exploited	LS2
Accessing Inspera exam outside of SEB	Exploited	Exploited	LS2
Allowing arbitrary process to be run in SEB	Exploited	Non-applicable	ET9
Launching multiple processes in SEB	Exploited	Non-applicable	ET9
Injecting text via USB device	Exploited	Exploited	ET15
Dictionary lookup on MacOS	Non-applicable	Exploited	ET12

Table 7.2: Summary of all attempted attacks.

adhere to. The following list was created by looking up a threat identifier in Table 6.1, and noting down which principles that can be used to mitigate the threat. In this section, we will use the abbreviations P1 for principle 1, and P2 for principle 2, and so on.

- ET9 breaks P3, P4, P5
- ET11 breaks P1, P2, P3, P4, P5
- ET12 breaks P3, P4, P5
- ET15 breaks P3, P4
- EI1 breaks P3
- EI4 breaks P6
- EI5 breaks P6
- LS2 breaks P1, P2

Summarizing the above list shows that all principles except P7 and P8 are not adhered to by SEB. As further emphasized in Section 6.7, P7 and P8 do not strictly have any corresponding threats that they can mitigate. They are instead used as guidelines for creating scalable and usable e-exam environments, which means that their conformity cannot be decided by crosschecking threats. Instead, the ability for different types of e-exam environments to adhere to P7 and P8 is further discussed in Chapter 9.

Lastly, P1-P6 are not adhered to by SEB, meaning that effort is required to make sure it does adhere to those principles. Doing so would increase the security of SEB as the attacks shown in this chapter would no longer be possible. In Chapter 8, we propose a secure design of SEB that attempts to follow all eight principles, mitigating all attacks found in this chapter.

8

A secure design for Safe Exam Browser

As discovered throughout the case study in Chapter 7, Safe Exam Browser contains fundamental design flaws that currently allow an examinee to bypass the software. This chapter will aim to propose a new design scheme that mitigates this flaw. We also show that this newly proposed design scheme follows the design principles presented in Chapter 6.

8.1 The fundamental design flaw in SEB

As shown in Chapter 7, the flaw in SEB is the fact that an examinee must know about the decryption password in order to start an exam. In other words, an examinee is required to manually enter the decryption password for the configuration file to be decrypted before an exam. Unfortunately, this means that an examinee can decrypt the configuration file outside of SEB as well, as long as they have the proper decryption procedure prepared beforehand, further described in Section 4.2. Since the decrypted configuration file is used to calculate the two security headers facilitated by SEB as seen in Section 4.5, an examinee can simply perform the same calculation to get the correct security headers and append them to their requests.

To conclude, since the examinee is given the decryption password, they can impersonate SEB trivially. In fact, no matter how many security headers that are implemented that rely on the same configuration file, the examinee will trivially be able to reconstruct them. This has previously been shown by Heintz in 2017 [13], where he reconstructed the `x-safeexambrowser-requesthash` present in the 2017 version. The developers have introduced more headers since then, which are still trivially reconstructable by an examinee with proper preparation, as shown by us in Subsection 7.5.2.

Therefore, it is feasible to conclude that the way SEB works today contains design flaws that allow an examinee to bypass the application and trivially impersonate SEB. The design of SEB needs to be reworked in order to successfully satisfy our proposed design principles in Chapter 6. In the following section, we will explain how *decoupling* the examinee and SEB will make the impersonation cryptographically infeasible. With this in mind, we will then propose a new design scheme for SEB, that successfully follows the design principles proposed in Chapter 6.

8.2 SEB and examinee separation

As mentioned in Section 8.1, the current underlying issue of SEB is the fact that the examinee has access to the decryption key of the configuration file. A design that allows decryption key access to SEB but not the examinee will prevent this issue, making it infeasible for an examinee to impersonate SEB.

However, allowing decryption key access to a user process like SEB, but not the user itself seems like an impossible challenge. SEB will likely be launched by an examinee's OS-level user, and then have the same permissions as that user. Thus, for SEB to have access to the decryption key, so must the user. This is because we are assuming a typical operating system where this decryption key would either need to be communicated or stored on the device as a file or similar. With the introduction of *Trusted Computing*, or *TC*, we have access to far more complex ways of performing and constraining cryptographic operations, aiding us in our goal of separating SEB and the examinee [63].

With Trusted Computing, we can convince a remote party that our host is in a certain state using *Remote Attestation*, encrypt and decrypt files using keys that are only accessible when our host is in a certain state with *Sealed Storage*, and much more. These two features are of interest to this design proposal since they would allow SEB to inform an LMS of its state, and constrain a key to only be accessible by SEB itself in a certain state.

8.3 Trusted computing

Trusted Computing (TC) is a security framework that aims to enhance the trustworthiness of computing systems and the data they handle [63]. At the core of trusted computing is the idea of using hardware-based security mechanisms, such as *Trusted Platform Modules* (TPMs) and secure enclaves, to protect against threats such as malware, viruses, and unauthorized access. Remote attestation and sealed storage are two key features of trusted computing that enable secure communication and integrity verification between different devices and provide a secure storage solution, respectively. In this section, we will discuss the concepts of remote attestation and sealed storage along with their role in enhancing the security and trustworthiness of e-exam environments.

8.3.1 Remote attestation

Remote attestation is a key feature in trusted computing that enables remote parties to verify the integrity of a host [63]. By having a dedicated *Hardware Security Module* (*HSM*) generate a certificate of, for example, the currently running software, or a generated key pair. This would allow a remote party to verify that unaltered software is currently running, or that the key pair has been generated according to some requirements. This would allow the remote party to be convinced that the host can be trusted. This will be important to our proposed design scheme as this

proof of correct state and integrity will be vital in the trust between an examinee's e-exam environment and an LMS.

8.3.2 Sealed storage

With HSMs, it is possible to create and constrain public/private key pairs such that only the HSM is able to access the keys when it is in a certain state [63]. If these keys would be used to encrypt certain files, this also implies that only the HSM is able to decrypt the files, given that it is in the correct state. Such state constraints can depend on the running software, entered credentials, configurations or similar. Additionally, the private key is stored inside the HSM, inaccessible by the examinee, while the public key can be distributed to any party. This means that encryption and decryption can only happen once a host is e.g. running a specific software and has reached a certain point throughout execution. This will be essential to our proposed design scheme, as this will allow us to make sure that an examinee is running a trusted piece of software (in this case SEB), and that the private key used for decryption is not accessible by the examinee. This is vital to the decoupling of SEB and the examinee, explained earlier in Section 8.2.

The TPM 2.0 chip [64], a common HSM, has very useful functionality for constraining a generated key pair with hardware components. The TPM 2.0 chip has *Platform Configuration Registers* (PCRs) which are hardware registers that always hold a current hash value. This hash value can never be written to directly, instead, it is the result of a chain of hashes. The value of a PCR can be updated by *extending* it with some value. The resulting value inside the PCR after the extension will be the output hash value of the previous hash value with the extended value appended to it. PCRs can be configured to depend on developer-supplied data, or the instruction sequence that is executed. Generated keys can be constrained to only be accessible and usable once a PCR holds a specific value. This unlocks the possibility to ensure that a certain sequence of instructions have executed on a host before allowing access to a key, which is essential to our design proposal.

8.4 Design proposal

Here we propose a secure design of Safe Exam Browser. With this proposal, we hope to contribute to the fundamental purpose of Safe Exam Browser, to make it harder for examinees to cheat in e-exams. Because of time constraints, no actual implementation of this proposed design is included. Instead, a detailed overview of the design is given to allow future authors or developers to freely interpret and construct a working implementation. Finally, the proposal aims to make sure that both the installation of SEB and starting an exam in SEB should be as easy as possible. The proposal, therefore, does not add any additional steps to installing or starting an exam with Safe Exam Browser from the perspective of an examinee.

8.4.1 Overview

The design consists of two procedures, one for the installation, and one for providing configuration files to SEB. These two procedures make sure that the running version of SEB is a trusted version, and that it has not been tampered with. It also keeps SEB and the examinee cryptographically separate, so that an examinee is not able to decrypt configuration files outside of SEB. This means that SEB is the only process that is able to decrypt the configuration files, making SEB the only process that can construct the proper security headers to provide proof that it is being used.

A simplified overview of the installation procedure for the design proposal is shown in Figure 8.1. The installation procedure begins at step 1, where an examinee would click a download link for the installation executable of SEB, while being authenticated to the LMS. The LMS then proceeds to request an installation executable from SEB Server specifically made for the authenticated examinee. The created installation executable is then sent to the requesting examinee, who launches the installation on their machine. During the installation, a private/public key pair will be created inside of the HSM to be used for sealed storage (further described in Subsection 8.3.2), where the public key is shared with SEB Server during the Remote Attestation process. These keys should be bound to SEB, via e.g. PCR constraints described in Subsection 8.3.2. If SEB Server successfully validates the attestation, the public key will be stored by SEB Server for future use when sealing configuration files. Finally, the installation executable is notified about whether or not the installation was successful or not.

It is important to note that the communication between the LMS and SEB Server is assumed to be trusted, as their communication otherwise might be interceptable by a malicious examinee. This kind of setup would likely require that an institution would host their own instance of SEB Server to be used for all e-exams.

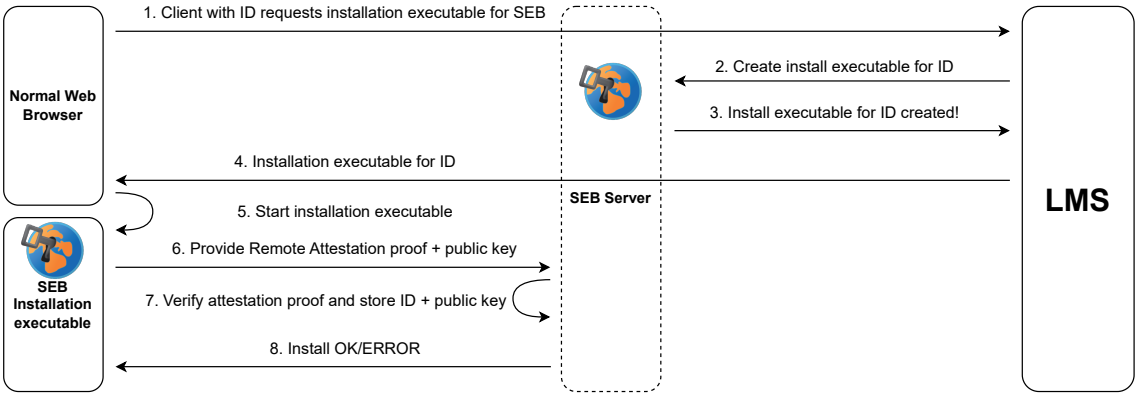

Figure 8.1: Overview of the installation procedure for the design proposal.

The second procedure introduced in this design proposal is the one for providing configuration files to examinees, shown in Figure 8.2. The examinee will provide authentication to their LMS (login credentials or similar), and request a configuration file for their exam. This configuration file should then be downloaded and applied to their running Safe Exam Browser process. This procedure is identical in

the perspective of an examinee, making sure that the design change does not affect the usability of the environment. One could even consider the usability to have been improved, as the password required to decrypt the configuration file has been completely removed, making the starting of an exam a very seamless experience.

The difference in our design proposal compared to the current design lies in the encryption used for configuration files. As steps 3-5 reveal, SEB Server will seal the configuration file using the requesting examinee's public key, meaning that only the examinee's Safe Exam Browser installation running in a trusted state will be able to unseal the configuration file. This is due to the keys being created during the installation process described in Figure 8.1, where they were bound to SEB itself. No examinee is able to unseal the configuration file outside of SEB.

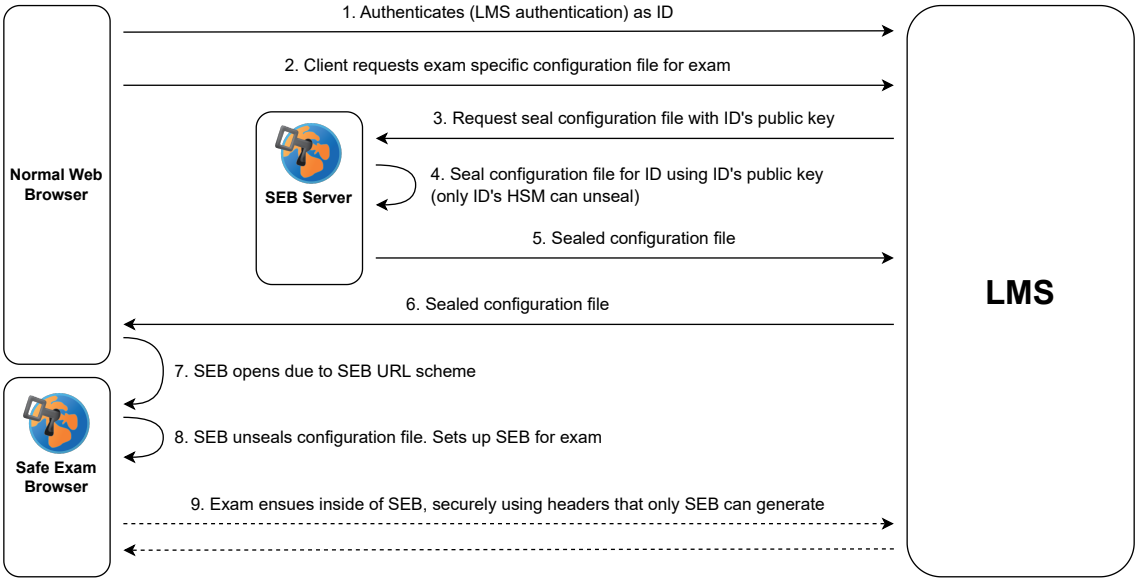

Figure 8.2: Overview of the procedure of providing configuration files using sealed storage.

8.4.2 Extending SEB Server

As Figure 8.1 and Figure 8.2 show, part of the design proposal requires the LMS to communicate with SEB Server. Currently, SEB Server is not required for exams that use SEB, it is instead optional and provides some additional functionality that is further described in Section 4.7. Our design proposal would require some added functionality to SEB Server to account for the creation of installation executables, Remote Attestation verification, and storing public keys for examinees.

Installation executables Installations need to be specific to examinees, meaning that only a specific examinee can use their installation for writing exams. This is because SEB Server needs to know which public key to use when sealing a configuration file, which is determined by the LMS providing the examinee's ID, as shown in steps 3-5 in Figure 8.2. Because of this, an installation needs to be individu-

ally tailored to an examinee with a corresponding ID, such that during steps 6-8 in Figure 8.1, the ID can be provided as part of the remote attestation procedure.

Practically, the LMS-specific ID for an examinee does not necessarily have to be part of the installation executable, only some kind of identifier. This identifier should be hard to guess so that you cannot trivially modify the installation executable before running it to install as a different examinee. An example of how to do this could be to randomly generate identifiers that are added to the installation executable, and map those random identifiers to actual LMS-specific examinee IDs. If the installation executable is never run within a certain time frame, the mapping can be timed out, forcing an examinee to have to re-download a fresh installation executable instead.

Remote attestation verification As described earlier, the installation executable will need to utilize remote attestation to prove the integrity of the installation, as well as the validity of the created public/private key pair. This corresponds to steps 6-8 in Figure 8.1. The remote attestation procedure will require an HSM to create a certificate of the executing installer, as well as the installed files. This certificate can then be signed by the root key of an HSM to prove its integrity and validity. SEB Server will simply check the signature and use the result of the check to decide if the installation can be trusted or not.

To prove the validity of the created public/private key pair (created in step 5 of Figure 8.1), a similar procedure can be followed. As introduced in Subsection 8.3.1, a signature could be created of the public key generated during the installation process, signed by the root key of an HSM. The signature can then be sent along with the public key to SEB Server. After this, SEB Server can check the signature to ensure that there is a corresponding private key stored inside of an HSM.

Public key storage Finally, SEB Server will need to be able to store and map LMS-specific examinee IDs to their corresponding public keys. This could be implemented as some kind of two-column database with the LMS-specific ID as a primary key, assuming that LMS-specific IDs are unique.

8.4.3 Extending Safe Exam Browser

Our proposal introduces the use of sealed storage (further described in Subsection 8.3.2) in SEB, to make sure that only SEB is able to construct the security headers added to the network traffic. As described briefly in Subsection 8.4.1, a public/private key pair should be generated during the installation of SEB, which can be seen in step 6 in Figure 8.1. The private key will be stored securely inside an HSM, and the public key will have to be shared with SEB Server. During the generation of this key pair, a certain set of constraints will have to be set up to make sure that only SEB is able to use them, further described in Subsection 8.3.2.

SEB will still require the decrypted configuration file in order to construct the security headers appended to its network traffic. However, since the examinee will be unable to decrypt the configuration file, no examinee can construct the security

headers outside of SEB unless they in some way have access to the decrypted configuration file. It is therefore important to emphasize the significance of protecting the decrypted configuration file, as it will have to be stored in an LMS. No examinee should ever be able to access the decrypted configuration file, since that would lead to an examinee being able to impersonate SEB completely.

8.4.4 SEB and LMS communication

Currently, the only proof of usage that SEB provides to an LMS are the security headers, and our design proposal makes no changes to that. As previously explained, the issue does not lie in the headers, but rather in how easily they are constructible by examinees. Therefore, no actual changes need to be made to the communication between SEB and the LMS during an exam. In fact, even the validation technique for the security headers is still correct, as only the LMS and SEB (and SEB Server when requested) should have access to the decrypted configuration file.

However, even if SEB provides such proof, it is equally important that this proof is validated and verified by the LMS. Therefore, the LMS needs to include some header that states the verification. From the LMS perspective, this could be done by first validating the security headers with the correct procedure, and then responding with $H(\text{security headers} + \text{headers valid? 1 or 0})$, where H is a hash function. SEB could then compute the same header and compare them, in order to make sure that the LMS is correctly validating the headers. Additionally, SEB should challenge the LMS before letting an examinee start their exam by sending invalid requests and making sure that the LMS correctly responds with $H(\text{security headers} + 0)$, and then sending valid requests that make sure the LMS correctly responds with $H(\text{security headers} + 1)$.

The most important change is that configuration files now need to be sealed by SEB Server before being shared with examinees. The communication between the LMS and SEB Server needs to be protected and secured so that no other party but the LMS is able to request sealed configuration files or create installation executables. Parties other than the LMS should only be able to go through the Remote Attestation procedure to prove their installation.

8.4.5 Addressing Hardware Security Module availability and compatibility

Subsection 8.4.3 briefly introduces the TPM 2.0 chip [64], which is often found on machines that run Windows. Microsoft themselves even claim that computers that are less than five years old should have TPM 2.0 functionality[1]. Other machines, such as Mac laptops will typically not have a TPM 2.0 chip[2], but instead offer similar functionality through other similar hardware components.

[1] `https://support.microsoft.com/en-us/topic/what-is-tpm-705f241d-025d-4470-80c5-4feeb24fa1ee`

[2] `https://discussions.apple.com/thread/253640878`

Küppers et al. [22] argue that no assumptions can be made about the capabilities of an examinee's brought device, specifically meaning that we cannot assume that hardware components capable of assisting with remote attestation are present in an examinee's device. Because of this, they employ a software-based remote attestation approach instead, while still recognizing that a hardware-based approach would be more secure.

In 2021, one year after Küppers et. al publication in 2020, Windows 11's first version was released. Windows 11 requires a TPM 2.0-enabled device[3], which will likely lead to many laptop manufacturers including a TPM 2.0 chip in their machines to make sure that they support the newer operating system. Like Microsoft also claims, computers that are less than five years old are very likely to have a TPM 2.0 chip. We, therefore, argue that requiring a TPM 2.0-enabled device or device with similar functionality is now a reasonable requirement. The number of examinees that will have a device with the required functionality will likely only become greater in the near future. We are aware that introducing a TPM-based solution puts pressure on the examinees to have compatible devices, however, if one chooses to not use TPM it becomes easier to cheat. We argue that the advantages of this requirement outweighs the disadvantages.

The improved design could also be made as opt-in functionality for institutions or similar that want to make use of it, similar to how the current functionality of SEB Server is opt-in. This would allow institutions to transition into a more secure e-exam environment as the implementation becomes more mature, while the number of HSM-enabled devices grows too.

8.4.6 Design principle conformity

In this final subsection, we will address how this design proposal conforms to the proposed design principles in Chapter 6.

Principle 1 With the introduction of remote attestation during the installation procedure of SEB, and constraining keys to a trusted PCR value, an LMS can be convinced that an examinee is using a trusted environment. It will be infeasible for an examinee to successfully forge a trusted installation procedure, and to forge a trusted environment, as they both rely on cryptographically hard operations using hardware security modules.

Principle 2 SEB in its current state allows integration developers of LMSs to neglect proper validation of the security measures provided. This is addressed by the communication changes proposed in Subsection 8.4.4. By enforcing that an LMS must always include a hash in its responses that signifies whether or not the request is valid, SEB can always be certain that the LMS is validating its traffic. As noted as well, to make sure that an LMS is truly validating the headers, SEB will have to challenge the LMS as part of its start-up sequence. Without such a challenge, an

[3] `https://support.microsoft.com/en-us/windows/windows-11-system-requirements-86c11283-ea52-4782-9efd-7674389a7ba3`

LMS could always respond with $H(\text{security headers} + 1)$, which defeats the entire purpose of the validation.

Principle 3 As the principle states, it should require *extreme effort* to access prohibited materials during an exam, to make it fair. With the changes in this design proposal, it will be infeasible to *not* use a trusted version of SEB, forcing examinees to use the exact same environment. Less can be said about any parity between different operating systems, as this design proposal will require different implementations depending on the target operating system. This means that the responsibility lies on the developers to make sure that the different platforms that SEB is compatible with provide similar levels of difficulty in accessing prohibited materials.

Principle 4 SEB Server provides improvements in logging and monitoring functionality. With our design proposal, SEB Server's logging and monitoring functionality are made available to the users of SEB. There is difficulty in arguing whether SEB Server's logging can be considered *suitable*, but it is far easier to realize that the logging it does implement will make it much harder for examinees to cheat unnoticed. Because of this, we still would like to argue that with our design proposal implements suitable logging, adhering to principle 4.

Principle 5 Similar to the argument made for principle 4, the extensive logging and monitoring made available through SEB Server will increase the likelihood of noticing when an examinee performs an unauthorized action.

Principle 6 SEB is already an open-source e-exam environment, and our design proposal does not require anything to be closed-source. Both the installation procedure and procedure of decrypting configuration files with the use of a hardware security module can be made available. This will hopefully also increase the security of the solution in the long term since anyone can view the code and provide improvements.

Principle 7 SEB is a highly scalable e-exam environment as it allows many examinees to write an exam at once. There is no distribution of USB sticks or similar which would inhibit the scalability of the environment. The only way that our proposal could be seen as affecting the scalability of the environment is through the availability of hardware security modules. This concern has however been addressed in Subsection 8.4.5 where we argue that the availability of such modules is already high, and is only going to increase in the near future. Thus, our proposal does not extensively affect the scalability of the environment.

Principle 8 As described in Section 8.4, this proposal makes few changes to *how* SEB is used. In fact, the only change made from the perspective of an examinee is the removal of a password to decrypt the configuration file, arguably increasing the usability. Therefore, our proposal does not extensively affect the usability of the environment.

8.4.7 Threat mitigation

In this final section, we briefly discuss how our design proposal, in conjunction with some of the mitigations described in Chapter 7, is able to mitigate the found threats in Chapter 5.

As Table 8.1 shows, 8 out of the 9 severe threats are mitigated, whereas the last was not investigated due to time constraints. In total, 28 out of the total 38 threats are successfully mitigated, leaving 10 unmitigated. Out of the 10 unmitigated, 6 cannot be mitigated by an operating system, or software at all. The remaining 4 were not investigated due to time constraints.

It is also evident from Table 8.1 that our design proposal, which focuses on P1 and P2, is part of the mitigation for 18 out of the 28 mitigated threats. It is clear that the issue described in Section 8.1, which prompted this design proposal, was an underlying issue to many threats since a solution to the issue mitigated many of them. With the introduction of our design principles, we were able to propose a new design for Safe Exam Browser which focused primarily on P1 and P2, that mitigated many threats. Further design proposals that focus on the other design principles may have additional positive effects on the possible threats.

Threat ID	Mitigated	Mitigation note
ES1	✓	Include DNS in PCR constraints, log IP-addresses
ES2	✓	Log activity and crosscheck with invigilator's attendance
ES3	✓	SEB does not allow such background processes, see Chapter 4
ES4	✓	Include hardware fingerprinting in remote attestation procedure, can then be validated by SEB Server, see Subsection 8.4.2
ES5	✓	Configure SEB Server to only be accessible from exam network, see Subsection 7.3.1
ET1	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
ET2	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
ET3	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
ET4	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
ET5	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
ET6	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3

Table 8.1 continued from previous page

Threat ID	Mitigated	Mitigation note
ET7	✓	Utilize HSMs to confirm valid OS, or use extensive logging of the results of OS-level actions
ET8	✓	Include firewall configurations in PCR constraints
ET9	✓	Include signatures of allowed third-party applications, see Subsection 7.6.4
ET10	✓	SEB already polls started processes in the background, stopping disallowed processes from launching, see Subsection 4.2.1
ET11	✓	Design proposal P1 and P2 conformity, also see note on ES1
ET12		Further investigation required
ET13		Further investigation required
ET14		Cannot be mitigated by OS or software
ET15	✓	Limit amount of connected keyboards, see Subsection 7.7.1
ET16		Cannot be mitigated by software [3]
ER1	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
EI1	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
EI2	✓	SEB is an open-source project
EI3	✓	SEB is an open-source project
EI4	✓	Design proposal P1 and P2 conformity uses hard cryptographic techniques, see Subsection 8.4.3
EI5	✓	Design proposal P1 and P2 conformity uses hard cryptographic techniques, see Subsection 8.4.3
LS1	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
LS2	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
LS3	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
XS1		Cannot be mitigated by OS or software
XS2		Cannot be mitigated by OS or software

Table 8.1 continued from previous page

Threat ID	Mitigated	Mitigation note
XS3		Cannot be mitigated by OS or software
XI1	✓	SEB disallows common third-party communication channels e.g. applications with chat functionality, see Subsection 4.2.1
XI2		Further investigation required
XI3	✓	Design proposal P1 and P2 conformity, see Subsection 8.4.3
XI4		Further investigation required
XI5		Cannot be mitigated by OS or software

Table 8.1: A summary of all threats and whether they are mitigated or not, with a short note describing the mitigation, or why no mitigation is shown.

9

Assessing the optimal environment

As previously presented in Chapter 3, there are two types of e-exam environments: software-based and OS-based. The question is whether both types could be made equally secure when following our design principles presented in Chapter 6, or if there exist threats that one of them cannot protect itself from. This chapter answers this question, which is related to **RQ3** of this thesis.

9.1 Trustworthiness comparison

The first pillar of our design principles, trustworthiness, considers both Principle 1 and Principle 2 (as seen in Section 6.1). Both of the two principles revolve around *trust*, which refers to the trust between the e-exam environment and its corresponding Learning Management System (LMS).

As presented earlier by us in Chapter 8, it is indeed possible to make a software-based e-exam environment meet Principle 1, by utilizing Trusted Computing. The same possibility exists for an OS-based e-exam environment, it could utilize Trusted Computing in order to prove that an examinee is using the correct, *trusted* OS.

Similarly, for Principle 2, our proposal in Chapter 8 addresses how to successfully meet the goals of the principle by utilizing a challenge-based authentication scheme for a software-based e-exam environment (in this case SEB). The same scheme could be used in an OS-based e-exam environment. The OS-based environment could use a constrained key pair, created via the Trusted Computing Remote Attestation process, to provide a verifiable signature of the request to the LMS. To meet the goals of Principle 2, the LMS could include a header similar to the one proposed in Subsection 8.4.4 which would allow the OS to challenge the LMS during a start-up sequence. This challenge-based authentication scheme would allow the OS to be confident that the LMS is correctly verifying its requests.

9.2 Fairness comparison

Principle 3, presented under the Fairness pillar in Section 6.2 states that "an e-exam environment *must* give all examinees the same level of access to materials". Although all found threats are not directly correlated with fairness, they all have

the potential to lead to an examinee cheating. Íf an examinee is able to cheat, they will have broken fairness, since they have gained an advantage over other examinees.

The challenging part of a software-based e-exam environment is assuring equal security between different operating systems. As mentioned previously, no operating system should give an examinee the upper hand. We believe that it is feasible to fulfill this goal, however, one must make sure to mitigate the found threats on all types of platforms. An OS-based e-exam environment does not suffer from this challenge, since it relies on its own dedicated operating system where everything takes place.

Hietanen [3] compares the security between a software-based approach versus an OS-based approach. The results show that the software-based approach is vulnerable to far more threats than the OS-based approach. The result is quite expected, since the software-based e-exam environment run within the examinees *own* OS, meaning that there is a high chance that an examinee may have tampered with either the OS or the software before the exam. However, as mentioned in the section above, a software-based e-exam environment can mitigate tampering-related threats by utilizing Trusted Computing.

The largest threat to a software-based e-exam environment is the fact that an examinee can modify the underlying OS or hardware. Some modifications are hard, if not impossible, to counter using a software-based approach. An example of such a threat is the chipset-related hardware attack (ET16) presented in Table 5.5, which is impossible to mitigate using a software-based approach. This shows that the software-based e-exam environment is vulnerable to more threats compared to an OS-based e-exam environment. As discussed, there exist some threats that are *impossible* to mitigate using a software-based approach. However, as presented in Section 6.2, Principle 3 states that all examinees should have the same level of access to materials. This means that it should require *extreme* efforts in order to access prohibited materials. We argue that the aforementioned threats are of this extreme nature, meaning that it is generally unfeasible for an examinee to utilize these types of threats. This can also be seen in Table 5.5, where ET16 is marked as non-severe. Therefore, we still believe that a software-based e-exam environment can be made equally secure as its OS-based counterpart, in terms of fairness.

9.3 Supervision comparison

Supervision, as presented in Section 6.3, consists of two principles: Principle 4 and Principle 5.

Both software-based and OS-based e-exam environments can fulfill Principle 4, which states that “an e-exam environment *must* implement suitable logging”. It may be that a software-based environment needs to log more information compared to the OS-based environment since it runs on an examinee’s untrusted OS. However, as discussed in Section 3.4, GDPR may be an issue preventing the logging of certain data on an examinee’s personal computer. Since an OS-based environment is completely isolated, meaning that the examinee is no longer running their usual OS,

it may be easier to adhere to these GDPR rules. This may be more challenging for the software-based environment, since logging of certain data may contain personal info.

Principle 5, "an e-exam environment must notice when an examinee performs an unauthorized action", is a bit trickier. Since the principle considers unauthorized actions, this includes every action that is not in line with the aim of the EEE. As mentioned in Section 9.2, hardware/OS modification threats are sometimes impossible for a software-based approach to mitigate, and the same goes for noticing them. Therefore, it might be that the software-based approach has a harder time noticing *all* unauthorized actions.

9.4 Transparency comparison

Principle 6 related to the transparency pillar (see Section 6.4) states that "an e-exam environment *must not* rely on security through obscurity". Both types of e-exam environments can definitely choose an open-source approach, effectively following Principle 6. Since both types can utilize Trusted Computing, as shown in Section 9.1, this implies that no examinee is able to modify the environments. Therefore, the only downside of an open-source approach is that examinees may have an easier time spotting flaws in the environment likely leading to cheating opportunities. This has been discussed previously in the report in Section 6.4, where we highlight why an open-source approach still outweighs its counterpart in terms of benefits.

9.5 Scalability comparison

As mentioned multiple times in this thesis, scalability is important to ensure that an e-exam environment is capable of scaling to a large number of examinees. Principle 7 of our design principles presented in Section 6.5 considers the scalability aspect of an environment.

The first major comparison factor is the fact that an OS-based e-exam environment is dependent on USB sticks, in order to distribute the environment. This has a relatively large impact on the scalability of the environment, due to the cost and time it takes to prepare such USB sticks. A possible solution would be to make each examinee download the OS, however, this approach affects the usability of the environment (see Section 9.6) since the procedure is far more complicated for a less experienced person.

Another factor affecting scalability in OS-based environments is the fact that a USB stick has the possibility of being corrupt. In case an environment is built like described in Subsection 3.3.2, an examinee's answers are stored locally on a USB stick and later retrieved after the exam has finished. There is a slight possibility that this may result in the USB stick being corrupt, meaning that it is no longer possible to retrieve the answers. This has a huge impact on the scalability aspect of the environment since the loss of an examinee's answers will have a large effect

on the general trust of the environment. The same scenario cannot happen in a software-based environment, since data is most likely stored directly to an LMS.

This shows that the software-based e-exam environment is better in terms of scalability. Since an OS-based environment is often distributed through USB sticks, this has an impact on the scalability, as mentioned above. This also implies that a software-based environment is better suited to meet the goals of Principle 7.

9.6 Usability comparison

Along with scalability, usability is another important factor that we have to take into account. Principle 8 presented in Section 6.6 outlines the importance of usability in an e-exam environment.

When investigating different e-exam environments in Chapter 3, we discovered a few comments regarding the usability of OS-based environments. For example, examinees had difficulties using ExamOS (presented in Subsection 3.3.1) due to unfamiliarity with the operating system itself. It was also reported that there were examinees who had difficulties booting the OS from a USB stick, once again due to unfamiliarity. The same result was shown as part of the national e-exam project in Australia (see Subsection 3.3.2), where some examinees had trouble navigating the OS.

This shows that it is sometimes difficult for examinees to use an OS that they are unfamiliar with, especially during an exam where they have a strict time limit, which has an impact on the overall usability of the environment. However, it is important to outline that it may be that the problem only persists for the first few exams, meaning that examinees may start feeling more comfortable using the OS with time. Even though this may be the case, the software-based e-exam environment has the upper hand overall, due to the aforementioned issues presented with OS-based e-exam environments. In other words, a software-based e-exam environment is better suited to meet the goals of Principle 8.

10

Conclusion

In summary, this thesis researches how to secure Bring-Your-Own-Device (BYOD) e-exam environments. The research focuses on two distinct types of e-exam environments: OS-based and software-based. Through a comprehensive threat modeling process, we identify potential cheating-related threats in both types of environments. Building upon this threat assessment, we put forth a set of carefully formulated *design principles*, intended to guide developers in designing robust and secure e-exam environments. Additionally, a detailed case study is conducted on the e-exam environment utilized at Chalmers University of Technology, known as Safe Exam Browser (SEB). This investigation reveals major security vulnerabilities within SEB. To address these shortcomings, we propose a novel SEB design that effectively integrates our design principles, reinforcing the security measures of the system. Finally, a comparative analysis is performed to distinguish the security capabilities of OS-based and software-based e-exam environments. By adhering to our design principles, we examine whether both approaches can attain a comparable level of security.

In this paper, we worked with the following four main research questions. Each question has been answered in their respective sections of the work, as described in Section 1.6, and we will go over our main findings once again.

RQ1: *What cheating threats exist for e-exams, and how severe are these?*

Through our utilization of the Quantitative Threat Modeling Method (QTMM) in conjunction with the STRIDE framework, we thoroughly examine three crucial components of the e-exam system: the examinee, the E-Exam Environment (EEE), and the Learning Management System (LMS). Our analysis leads to the identification of numerous cheating-related threats, which we visualize using attack trees. Furthermore, we classify these threats based on severity, revealing a total of 38 threats, 9 of which we categorize as severe. This comprehensive assessment empowers us to gain valuable insights into the vulnerabilities present within the e-exam system, forming the basis for the design principles proposed in this thesis.

RQ2: *What is a set of common design principles for e-exam environments that, when followed, will ensure that the environment is secure and those cheating threats are mitigated?*

By utilizing the results found during the threat modeling process, we propose design principles that revolve around six pillars: Trustworthiness, Fairness, Supervision,

Transparency, Scalability, and Usability. We are putting forth eight principles that guide developers towards creating robust and secure e-exam environments. The design principles undergo testing as we propose a new design scheme for Safe Exam Browser. We demonstrate that by following the design principles, we successfully mitigate many of the previously identified cheating threats.

RQ3: *Will a software-based e-exam environment and an OS-based e-exam environment be able to achieve a similar level of security when following the design principles?*

Both environments exhibit advantages and disadvantages in relation to specific pillars. When considering fairness, it becomes apparent that the OS-based environment effectively mitigates a greater number of threats compared to its software-based counterpart. However, these threats are of an extreme nature, requiring significant effort from an examinee to successfully exploit them, thereby not falling into the category of severe threats. Of course, they are still threats, and someone may potentially use them to cheat.

Despite the OS-based e-exam environment's proficiency in mitigating a larger number of threats, it faces challenges in terms of scalability and usability. This indicates that if scalability and usability are deemed crucial, an OS-based approach may not be the optimal choice, even with its superior threat mitigation capabilities compared to the software-based alternative.

RQ4: *Does the e-exam environment at Chalmers University of Technology, Safe Exam Browser, satisfy the design principles and therefore mitigate the found cheating threats?*

During our case study of Safe Exam Browser, we discover significant security flaws within the application. In its current state, it is possible for an examinee to completely bypass the application. Furthermore, we demonstrate that six out of the eight total design principles are not satisfied by showcasing successful attacks that could be mitigated by following a corresponding principle. We suggest possible mitigations for the successful attacks, along with proposing a completely new design scheme for Safe Exam Browser that follows our proposed design principles.

10.1 Future work

The work done by us in this thesis has the possibility of being developed even further.

Implementing the new SEB design and assessing its security We proposed a new SEB design in Chapter 8. Future work could implement the design and perform a thorough penetration test on it to verify whether it mitigates the cheating threats we found in Chapter 7.

Further research on OS-based e-exam environments As highlighted in both Chapter 9 and Chapter 3, current OS-based e-exam environments suffer from scalability and usability issues. Future work could include conducting further research on such an environment, with the goal of developing a solution that is more scalable and usable.

Developing a new secure-by-design e-exam environment Possible future work could develop a new e-exam environment intended to be secure-by-design, by following our design principles proposed in Chapter 6.

Assessing the security of an OS-based e-exam environment Similar to the case study performed by us in Subsection 7.1.1, a similar study could be conducted on an OS-based e-exam environment. The security of OS-based e-exam environments has previously been examined by Hietanen [3], but a possible future work could be to examine the security of the environment with our principles in mind.

Bibliography

[1] G. Sindre and A. Vegendla, "E-exams versus paper exams: A comparative analysis of cheating-related security threats and countermeasures," in *Norwegian Information Security Conference (NISK)*, vol. 8, 2015, pp. 34–45.

[2] *Safe Exam Browser*, `https://safeexambrowser.org/news_en.html`, Accessed: 2022-11-24.

[3] J. Hietanen, "Security of electronic exams on students' devices - M.Sc Thesis," *Aalto University*, 2021.

[4] D. Deogun, D. B. Johnsson, and D. Sawano, *Secure by Design*. Manning Publications, 2018.

[5] M. Abadi and R. M. Needham, "Prudent engineering practice for cryptographic protocols," *IEEE Trans. Software Eng.*, vol. 22, no. 1, pp. 6–15, 1996.

[6] I. Bastys, F. Piessens, and A. Sabelfeld, "Prudent design principles for information flow control," in *PLAS@CCS*, ACM, 2018, pp. 17–23.

[7] A. Chirumamilla, "Analysis of security threats, requirements, and technologies in e-exam systems - Ph.D Thesis," *NTNU*, 2021.

[8] K.-P. Yee, "Aligning security and usability," *IEEE Security & Privacy*, vol. 2, no. 5, pp. 48–55, 2004.

[9] A. Vegendla and G. Sindre, "Mitigation of cheating in online exams: Strengths and limitations of biometric authentication," in *Biometric authentication in online learning environments*, IGI Global, 2019, pp. 47–68.

[10] P. Engebretson, *The basics of hacking and penetration testing: ethical hacking and penetration testing made easy*. Elsevier, 2013.

[11] P. Karpati, A. L. Opdahl, and G. Sindre, "Harm: Hacker attack representation method," in *Software and Data Technologies: 5th International Conference, ICSOFT 2010, Athens, Greece, July 22-24, 2010. Revised Selected Papers 5*, Springer, 2013, pp. 156–175.

[12] A. Vegendla, T. M. Søgaard, and G. Sindre, "Extending HARM to make test cases for penetration testing," in *CAiSE Workshops*, ser. Lecture Notes in Business Information Processing, vol. 249, Springer, 2016, pp. 254–265.

[13] A. Heintz, "Cheating at digital exams-vulnerabilities and countermeasures," M.S. thesis, NTNU, 2017.

[14] T. M. Søgaard, "Mitigation of cheating threats in digital byod exams," M.S. thesis, NTNU, 2016.

[15] *Safe Exam Browser 3.0.0*, `https://github.com/SafeExamBrowser/seb-win-refactoring/releases/tag/3.0.0`, Accessed: 2022-12-12.

[16] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.
[17] G. Conole and B. Warburton, "A review of computer-assisted assessment," *ALT-J*, vol. 13, no. 1, pp. 17–31, 2005.
[18] *About SEB server - SEB Server documentation*, https://seb-server.readthedocs.io/en/latest/overview.html, Accessed: 2023-04-20.
[19] *The secure exam platform*, https://www.digiexam.com/, Accessed: 2022-11-27.
[20] *High security and privacy with our proprietary technology | Digiexam*, https://www.digiexam.com/enterprise-security, Accessed: 2023-04-21.
[21] B. Küppers, M. Politze, R. Zameitat, F. Kerber, and U. Schroeder, "Practical security for electronic examinations on students devices," in *Intelligent Computing: Proceedings of the 2018 Computing Conference, Volume 2*, Springer, 2019, pp. 290–306.
[22] B. Küppers, T. Eifert, R. Zameitat, and U. Schroeder, "EA and BYOD: threat model and comparison to paper-based examinations," in *CSEDU (1)*, SCITEPRESS, 2020, pp. 495–502.
[23] G. Coker *et al.*, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, pp. 63–81, 2011.
[24] *ExamOS - GitHub*, https://github.com/konekoe, Accessed: 2023-04-21.
[25] J. Pagram, M. Cooper, H. Jin, and A. Campbell, "Tales from the exam room: Trialing an e-exam system for computer education and design and technology students," *Education Sciences*, vol. 8, no. 4, p. 188, 2018.
[26] M. Hillier, "To type or handwrite: Students experience across six e-exam trials," *ascilite2015*, p. 143, 2015.
[27] *General Data Protection Regulation (GDPR)*, https://gdprinfo.eu/, Accessed: 2023-04-21.
[28] *General Data Protection Regulation (GDPR) - Article 17 - Right to erasure ('right to be forgotten')*, https://gdprinfo.eu/en-article-17, Accessed: 2023-05-11.
[29] *About Safe Exam Browser*, https://safeexambrowser.org/about_overview_en.html, Accessed: 2022-11-27.
[30] *Safe Exam Browser - GitHub*, https://github.com/SafeExamBrowser, Accessed: 2023-01-11.
[31] *Safe Exam Browser - Windows User Manual*, https://safeexambrowser.org/windows/win_usermanual_en.html, Accessed: 2023-02-06.
[32] *Safe Exam Browser - Mac User Manual*, https://safeexambrowser.org/macosx/mac_usermanual_en.html, Accessed: 2023-02-06.
[33] *Safe Exam Browser - Developer Documentation | Config File Format*, https://safeexambrowser.org/developer/seb-file-format.html, Accessed: 2023-02-22.
[34] M. Dworkin *et al.*, *Advanced encryption standard (aes)*, en, 2001-11-26 2001. DOI: https://doi.org/10.6028/NIST.FIPS.197.
[35] *RFC 2898 - PKCS #5: Password-Based Cryptography Specification, Version 2*, https://doi.org/10.17487%2FRFC2898, Accessed: 2023-02-22.

[36] *Safe Exam Browser - Developer File Format*, https://safeexambrowser.org/developer/seb-file-format.html, Accessed: 2023-03-09.

[37] *Safe Exam Browser - macOS User Manual - Permitted Processes*, https://safeexambrowser.org/macosx/mac_usermanual_en.html#ApplicationsPane, Accessed: 2023-03-10.

[38] *Safe exam browser: Developer - integration*, https://safeexambrowser.org/developer/seb-integration.html, Accessed: 2023-03-10.

[39] *Safe Exam Browser - Developer Information - Config Key*, https://safeexambrowser.org/developer/seb-config-key.html, Accessed: 2023-03-10.

[40] *Safe Exam Browser: Privacy Statement*, https://safeexambrowser.org/about_overview_en.html#privacy-statement, Accessed: 2023-04-20.

[41] *Threat Modeling | OWASP Foundation*, https://owasp.org/www-community/Threat_Modeling, Accessed: 2023-01-31.

[42] A. Shostack, *Threat modeling: Designing for security*. John Wiley & Sons, 2014, p. xxiii.

[43] B. Potteiger, G. Martins, and X. D. Koutsoukos, "Software and attack centric integrated threat modeling for quantitative risk assessment," in *HotSoS*, ACM, 2016, pp. 99–108.

[44] *Common Vulnerability Scoring System SIG*, https://www.first.org/cvss/, Accessed: 2023-01-27.

[45] W. S. Code, "By michael howard and david c," *LeBlanc, is a fairly comprehensive discussion of secure coding practices that is a good reference for understanding buffer overflows and many other types of vulnerabilities that can creep into software*, 2002.

[46] L. Kohnfelder and P. Garg, "The threats to our products," *Microsoft Interface*, 1999.

[47] A. Bagnato, B. Kordy, P. H. Meland, and P. Schweitzer, "Attribute decoration of attack-defense trees," *Int. J. Secur. Softw. Eng.*, vol. 3, no. 2, pp. 1–35, 2012.

[48] T. Sonderen, "A manual for attack trees," M.S. thesis, University of Twente, 2019.

[49] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero trust architecture," National Institute of Standards and Technology, Tech. Rep., 2020.

[50] *OWASP Application Security Verification Standard*, https://owasp.org/www-project-application-security-verification-standard/, version 4.0.3.

[51] S. Balfe and A. Mohammed, "Final fantasy - securing on-line gaming with trusted computing," in *Autonomic and Trusted Computing, 4th International Conference, ATC 2007, Hong Kong, China, July 11-13, 2007, Proceedings*, B. Xiao, L. T. Yang, J. Ma, C. Müller-Schloer, and Y. Hua, Eds., ser. Lecture Notes in Computer Science, vol. 4610, Springer, 2007, pp. 123–134. DOI: 10.1007/978-3-540-73547-2_15. [Online]. Available: https://doi.org/10.1007/978-3-540-73547-2%5C_15.

[52] Secret Club, *The nadir of surveillance (Den Digitale Prøvevagt)*, https://secret.club/2019/03/07/exam-surveillance.html, Accessed: 2023-04-27.

[53] Version2, *Gymnasieelev piller Den Digitale Prøvevagt fra hinanden finder 90er-kryptering*, https://www.version2.dk/artikel/gymnasieelev-pill

er-den-digitale-proevevagt-fra-hinanden-finder-90er-kryptering, Accessed: 2023-04-27.

[54] R. Kainda, I. Flechais, and A. Roscoe, "Security and usability: Analysis and evaluation," in *2010 international conference on availability, reliability and security*, IEEE, 2010, pp. 275–282.

[55] M. A. Sasse, M. Smith, C. Herley, H. Lipford, and K. Vaniea, "Debunking security-usability tradeoff myths," *IEEE Security & Privacy*, vol. 14, no. 5, pp. 33–39, 2016.

[56] S. Acharya and V. Pandya, "Bridge between black box and white box–gray box testing technique," *International Journal of Electronics and Computer Science Engineering*, vol. 2, no. 1, pp. 175–185, 2012.

[57] *Safe Exam Browser - Developer Documentation*, https://safeexambrowser.org/developer/, Accessed: 2023-03-12.

[58] E. Rescorla and A. M. Schiffman, *The Secure HyperText Transfer Protocol*, RFC 2660, Aug. 1999. DOI: 10.17487/RFC2660. [Online]. Available: https://www.rfc-editor.org/info/rfc2660.

[59] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, RFC 8446, Aug. 2018. DOI: 10.17487/RFC8446. [Online]. Available: https://www.rfc-editor.org/info/rfc8446.

[60] *Chrome Root Program Policy, Version 1.4*, https://www.chromium.org/Home/chromium-security/root-ca-policy/, Accessed: 2023-03-13.

[61] *Brute Forcing Passwords - Cocoaphony*, https://robnapier.net/brute-forcing-passwords, Accessed: 2023-03-21.

[62] *Look up words on Mac | Issue #270 SafeExamBrowser/seb-mac*, https://github.com/SafeExamBrowser/seb-mac/issues/270#issuecomment-1461785663, Accessed: 2023-04-12.

[63] C. Mitchell, *Trusted computing*. Iet, 2005, vol. 6.

[64] *TPM 2.0 Library | Trusted Computing Group*, https://trustedcomputinggroup.org/resource/tpm-library-specification/, Accessed: 2023-05-10.