CHALMERS

GÖTEBORGS UNIVERSITET

# Anpassad applikation för låsning av Windows-enheter baserad på Safe Exam Browser

Examensarbete inom högskoleingenjörsprogrammet i datateknik

Oliver Dahlén
John Saltin

**INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK**
CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2024
www.chalmers.se

Examensarbete 2024

# Anpassad applikation för låsning av Windows-enheter baserad på Safe Exam Browser

Oliver Dahlén
John Saltin

Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
Göteborg 2024

Anpassad applikation för låsning av Windows-enheter
baserad på Safe Exam Browser

Oliver Dahlén
John Saltin



Handledare:
Nicholas Smallbone, Institutionen för data- och informationsteknik
Anonym handledare på anonymt företag

Examinator:
Jonas Almström Duregård, Institutionen för data- och informationsteknik

Examensarbete 2024
Institutionen för data- och informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg, Sverige
Telefon +46 31 772 1000

Göteborg 2024

# Sammanfattning

Digitala prov har blivit vanliga i dagens utbildningssystem. De erbjuder många fördelar jämfört med traditionella pappersbaserade prov. Dessa inkluderar bland annat minskad administrativ börda för lärare, möjlighet att enklare redigera text samt automatisk och anonym rättning. Dock finns även nackdelar med digitala prov såsom ökad risk för fusk och kravet på skolor att ta fram eller upphandla en användarvänlig och stabil provplattform. Ett företag som i denna rapport är anonymt arbetar med att utveckla en provplattform till många olika operativsystem. Till Windows använder företaget i dagsläget en omodifierad version av Safe Exam Browser (SEB). Företaget har skapat en lista med förbättringar till en egen SEB-version. Syftet med förbättringarna är att göra applikationen mer användarvänlig både för användare och utvecklare samt att förstärka fuskpreventionsmetoder. I denna rapport beskrivs hur arbetet med att utveckla förbättringarna gått till. De fuskpreventionsmetoder som utforskats är primärt detektion av virtuella maskiner. Två metoder för att försöka göra detta har implementerats. Dessa använder sig av Trusted Platform Module-enheten och grafikprocessorprestandaräknare i Windows. Pålitligheten att detektera virtuella maskiner med dessa metoder är inte helt fastställd och bör undersökas ytterligare.

Nyckelord: digitala prov, Safe Exam Browser, Trusted Platform Module, virtuell maskin detektion

# Abstract

Digital examinations have become commonplace in today's education system. They offer many advantages over traditional paper-based examinations. These include, amongst other things, reduced administrative burden for teachers, the ability to edit text easier as well as automatic and anonymous grading. However, there are disadvantages with digital examinations such as an increased risk of cheating and the demand for schools to procure or develop a user-friendly and stable examination platform. A company that is anonymous in this report are developing an examination platform for many different operating systems. For Windows the company uses an unmodified version of Safe Exam Browser (SEB). The company has put together a list of improvements for their own SEB version. The purpose of the improvements is to make the application more user-friendly for both users and developers, and to strengthen cheat prevention. This report describes the process of developing these improvements. The cheat prevention methods that have been explored are primarily detection of virtual machines. Two methods to try to accomplish this have been implemented. These methods use the Trusted Platform Module device and graphical memory performance counter in Windows. The reliability of using these methods to detect virtual machines is not established and needs further investigation.

Keywords: Digital examinations, Safe Exam Browser, Trusted Platform Module, virtual machine detection

# Akronymer

En lista av akronymer som används i texten sorterad i alfabetisk ordning:

| | |
|---|---|
| API | Application programming interface |
| EK | Endorsement key |
| ETH Zürich | Federal Institute of Technology Zürich |
| GPU | Graphics processing unit |
| MPL 2.0 | Mozilla public license 2.0 |
| SEB | Safe Exam Browser |
| TPM | Trusted platform module |
| VM | Virtuell maskin |
| WDDM | Windows Display Driver Model |

# Ordlista

En lista på ord och termer som förekommer i rapporten:

**Attackerare:** En person som utnyttjar sårbarheter i datasystem för sina egna ändamål

**Attackerarmodell:** En beskrivning på en attackerares förmågor och begränsningar

**Attestering:** En metod inom kryptografi som är vanlig för att validera en enhets identitet

**Företag(et):** Det anonyma företag som projektmedlemmarna samarbetat med

**GPU-passthrough:** En metod för att ge en virtuell maskin tillgång till datorns grafikkort

**Mozilla public license:** En mjukvarulicens som används till öppen källkod

**Named Pipes:** Ett sätt för processer i ett operativsystem att kommunicera med varandra

**Projekt(et):** Examensarbetet som utförts och beskrivs i denna rapport

**Projektmedlemmar:** De personer som utfört examensarbetet

**Safe Exam Browser:** Ett mjukvaruprojekt med öppen källkod som ger möjlighet att skriva prov i en digital miljö

**Safe Exam Browsers röda låsskärm:** En röd skärm som låser datorn när SEB detekterar misstänkt aktivitet på datorn

**TPM-passthrough:** En metod för att ge en virtuell maskin tillgång till datorns TPM-enhet

**Trusted platform module:** En kryptografisk enhet som bland annat används till att spara och skydda känslig information

**Virtuell maskin:** En isolerad, virtuell miljö som efterliknar fysisk datorhårdvara och låter dess användare köra flera operativsystem samtidigt

# Innehållsförteckning

# 1. Inledning

## 1.1 Bakgrund

Det är idag vanligt att prov i skolan utförs digitalt. För att detta ska kunna ske på ett säkert och rättvist sätt behöver man speciell programvara för att möjliggöra skrivning av proven och även för att förhindra fusk. Detta gäller i synnerhet prov där elever skriver provet på sin egen enhet. Fusk är vanligt förekommande i den svenska skolan. I en undersökning från 2003 uppgav 70% av gymnasieelever att de har fuskat minst en gång på skriftliga prov [1]. På grund av detta är en avgörande aspekt av provprogramvara dess förmåga att försvåra fusk. Det främsta verktyget för att åstadkomma detta är en lockdownfunktion som låser enheten och förhindrar eleven att lämna provmiljön och använda andra program under tiden provet skrivs.

En vanligt förekommande plattform som erbjuder en digital provmiljö är Safe Exam Browser (SEB). SEB har öppen källkod och släpps under licensen MPL 2.0 [2]. Plattformen utvecklas av Federal Institute of Technology Zürich (ETH Zürich) [3] och finansieras av SEB Alliance. Medlemmarna i SEB Alliance är företag och institutioner [4]. Idag finns ett 20-tal medlemmar vars medlemskap tillåter dem att vara med och bestämma riktningen som plattformen utvecklas i.

SEB har utvecklats sedan 2009 och är i dagsläget en provplattform med mycket funktionalitet. Det finns klienter till Windows, macOS och iOS. Utöver de olika klienterna finns tre valfria verktyg; en serverapplikation, ett verifieringsverktyg och ett konfigurationsverktyg. Serverapplikationen underlättar storskaligt användande av SEB. Detta genom att bland annat tillåta centraliserad konfigurering av klienterna vid provtillfällen och öka säkerheten genom att tillåta övervakning under provskrivningar. Verifieringsverktyg laddas in på en USB-sticka. Det används bland annat när SEB utlöst en speciell typ av fuskvarning under ett provtillfälle. Provvakten går då fram och använder USB-stickan för att verifiera integriteten i applikationens programkod. Konfigurationsverktyget ger möjlighet att ändra inställningar i provmiljön genom ett grafiskt gränssnitt.

Fokus i detta projekt ligger på Windowsversionen av SEB. En lista av önskade förändringar togs fram av ett företag som projektmedlemmarna fått kontakt med. Ur denna lista valdes i samråd med företaget fem förändringar ut som bedömdes utgöra en lagom mängd arbete. En av förändringarna omfattar säkerhetsfunktioner och på grund av detta önskade företaget att förbli anonyma i rapporten, vilket naturligtvis respekteras av projektmedlemmarna. Företaget kommer därför inte att benämnas vid namn i rapporten och vissa detaljer kommer att utelämnas för att säkerställa att de förblir anonyma.

Företaget utvecklar en plattform som låter lärare skapa och genomföra prov med sina elever. För att åstadkomma fullständig nedlåsning av Windowsenheter används i dagsläget SEB vilket ger upphov till en del problem. Ett problem är att företaget inte har kontroll över utvecklingen av applikationen och därmed får problem med integrationen, både vad gäller funktion och säkerhet. Ett annat problem är att eftersom applikationen har öppen källkod

kan vem som helst gå in och hitta svagheter och säkerhetsluckor i koden, eller göra modifieringar och försöka använda en egenbyggd version vid prov.

## 1.2 Syfte

Projektets syfte är att undersöka på vilka sätt man kan modifiera och förbättra företagets Windows-applikation. Detta görs inom tre huvudområden som kommer att operationaliseras nedan; förbättrad användarvänlighet för provtagare, förbättrad funktionalitet för utvecklare och förbättring av säkerhetsmekanismer för fuskdetektion. Genom att projektet undersöker ett faktiskt företags nuvarande problem med sin programvara vill projektmedlemmarna fördjupa sin förståelse kring de säkerhetsfunktioner som hårdvara och operativsystem erbjuder för att skydda sina användare.

## 1.3 Mål

Projektets huvudsakliga målsättning är att förbättra företagets provplattform genom att modifiera SEB:s funktionalitet. Modifikationerna omfattar säkerhetsfunktioner och användarvänlighet för både provtagare och företagets utvecklare. För att uppnå detta skapades i samråd med företaget en kravspecifikation. Punkterna i kravspecifikationen sattes upp som målsättningar för projektet och listas nedan. I rapporten kommer kraven att benämnas som delmål.

**Delmål 1:** Ge möjligheten att stänga av en säkerhetsfunktion i SEB som utlöser en röd låsskärm för att istället skicka den informationen till företagets servrar, så att beslut kan tas om huruvida eleven får fortsätta provet eller inte.

**Delmål 2:** Skapa ett sätt att dynamiskt ladda konfigurationsfiler genom kommunikation med företagets servrar, istället för att ladda in lokala konfigurationsfiler vid uppstart.

**Delmål 3:** Skicka errorloggar som SEB genererar vid körningsfel till företagets servrar, istället för att spara dessa i lokala filer.

**Delmål 4:** Möjliggör uppstart av SEB manuellt av eleven, istället för att applikationen startas via företagets webbplats. När applikationen startas manuellt ska en API-förfrågan skickas till företagets servrar där en konfigurationsfil ska erhållas.

**Delmål 5:** Lägga till och hitta ytterligare sätt att detektera att SEB körs i en virtuell maskin, samt utforska och utveckla metoder för att se om applikationen är modifierad.

Ytterligare ett krav från företaget är att ändringarna i applikationen inte ska bryta mot licensen MPL 2.0 som SEB släpps under. Utöver detta ska applikationen utvecklas på ett sätt

som gör det så enkelt som möjligt att synkronisera företagets kodbas med uppdateringar i SEB:s kodbas.

## 1.4 Avgränsningar

Provfusk kan förekomma i båda fysiska och digitala former. Denna rapport kommer endast att fokusera på den digitala provmiljön och ingen hänsyn kommer tas till fysiska fuskmetoder såsom fusklappar eller sekundära enheter.

En ytterligare avgränsning är att projektet bara kommer att fokusera på utveckling i operativsystemet Windows. Primärt fokus är att det ska fungera i Windows 11. Om möjligt ska det även fungera i Windows 10. Bakåtkompatibilitet på tidigare Windowsversioner är önskvärt men ej nödvändigt.

Utvecklingen är begränsad till SEB. Där det kan komma att krävas integration med företagets plattform kommer företaget att bistå med hjälp att implementera nödvändig funktionalitet.

Vad gäller delmål 5 begränsas säkerhetsfunktionerna till en attackerare som kan beskrivas som en tekniskt kunnig provtagare. Med andra ord en person som vill fuska på ett prov och har tillräckligt med kunskap för att modifiera programvara för att undvika fuskdetektion.

Avsikten är inte att projektets slutprodukt ska vara leveransklar. Det kommer att krävas ytterligare utveckling och testning innan applikationen kan användas i produktion.

# 2. Metod

## 2.1 Utvecklingsmiljö och programspråk

För att hantera och utveckla koden på ett enkelt sätt kommer Visual Studio 2022 att användas. Detta eftersom SEB utvecklas i Visual Studio och det är en väldokumenterad utvecklingsmiljö. Det primära programmeringsspråket för utvecklingen av applikationen kommer att vara C#. Språket möjliggör att snabbt utveckla funktioner till Windows och valet faller sig naturligt eftersom SEB är byggt med C#.

## 2.2 Inlärning av programspråk

Projektmedlemmarna har tidigare aldrig använt C# och kommer att behöva lära sig detta. De har sedan tidigare använt C och Java och har därmed en förståelse för syntaxen och hur objektorienterade språk fungerar. De nya koncepten som C# erbjuder kommer studeras genom kodkurser och instruktionsvideor online.

## 2.3 Programmeringsstrategi

För att underlätta fortsatt utveckling av applikationen för företaget kommer man att försöka ändra så lite som möjligt i SEB:s befintliga kodbas. Ny funktionalitet och ändringar placeras istället om möjligt i nya filer. Det gör det enklare att synkronisera företagets kodbas med uppdateringar från SEB:s kodbas med så lite konflikter som möjligt och följer således företagets krav angående MPL 2.0.

## 2.4 Versionshantering

Hanteringen och sparandet av programkoden kommer att ske via versionshanteringssystemet Git och plattformen GitHub [5]. Då SEB också använder GitHub blir det enklare för företaget att synkronisera sin egen kodbas med uppdateringar från SEB:s kodbas.

## 2.5 Utvecklingsmodell

För utveckling kommer vattenfallsmodellen användas eftersom det finns tydligt uppsatta mål som sannolikt inte kommer förändras. Man övervägde även att använda sig av en agil arbetsmetodik, men beslutade emot detta då sprintmöten hade tagit för mycket tid i anspråk.

## 2.6 Teststrategi

För att säkerställa att programkoden fungerar kommer de befintliga testerna i SEB:s kodbas att bibehållas. Om de befintliga testerna slutar fungera under utvecklingens gång kommer projektmedlemmarna att återställa testernas funktionalitet. Det kommer även att skapas enhetstester för all ny funktionalitet.

För att testa de metoder som används för att detektera virtuella maskiner kommer ett antal olika mjukvaror att användas. Dessa är VMWare, Hyper-V, VirtualBox, QEMU.

## 2.7 Utvärdering av resultat

Om implementationerna i delmålen 1–4 ger den funktionalitet som beskrivs i respektive delmål och dessutom klarar de enhetstester som skapas kommer dessa att anses vara avklarade. Delmål 5 har en öppen problemställning och det är svårare att sätta ett tydligt slutmål. Resultatet kommer istället att utvärderas baserat på hur bra skyddet uppskattas vara mot attackerarmodellen som specificerats i avsnitt 1.4.

# 3. Teknisk bakgrund

Nedan beskrivs begrepp och koncept som förekommer i rapporten. Dessa inkluderar hur SEB är uppbyggd samt de relevanta tekniska elementen den använder. Olika tekniker och bakgrundsförståelse för att detektera virtuella maskiner kommer även att tas upp här.

## 3.1 Safe Exam Browser

SEB är en digital provplattform för att möjliggöra prov på en digital enhet, och samtidigt säkerställa att provtagare inte fuskar under proven. SEB utvecklas av ETH Zürich. Plattformen har öppen källkod och släpps under licensen MPL 2.0. Huvudapplikationens struktur är modulär och uppdelad i tre separata delar: Runtime, Client och Service. Dessa körs i tre helt egna processer och kommunicerar med varandra genom named pipes. Det finns även en Browser-process som inte är utvecklad av ETH Zürich och är fristående från resten av applikationen. En beskrivning på SEB:s struktur finns att hitta på [3].

Utöver huvudapplikationen finns tre andra verktyg som inkluderar en serverapplikation, ett verifieringsverktyg och ett konfigurationsverktyg. Dessa kan valfritt användas för att stödja huvudapplikationen så att den blir mer säker och användarvänlig.

### 3.1.1 Runtime

Denna process är den centrala delen av applikationen som sköter all kommunikation mellan de olika processerna. Den hanterar även att de andra processerna startas och att operativsystemet går in i ett låst läge där provtagaren inte kan lämna provmiljön för att komma åt otillåtna resurser på enheten.

### 3.1.2 Client

Client-processen är den delen av applikationen som sköter användargränssnittet och även integrerar webbläsaren. Client-processen övervakar även olika systemelement för att upptäcka försök till fusk. Detta inkluderar saker som registervärden och bakgrundsprocesser som kan vara en säkerhetsrisk.

### 3.1.3 Service

Service-processen används för att bistå de andra processerna med information som bara operativsystemets privilegierade administrationskonton har tillgång till. Processen tillhandahåller extra säkerhetsfunktioner i syfte att förhindra fusk. Eftersom man inte kan förutsätta att administrationsåtkomst finns, är den inte nödvändig för att SEB ska fungera på ett korrekt sätt. Det går således att välja om den ska användas eller inte.

### 3.1.4 Browser

Browser-processen har hand om webbläsarfunktionaliteten. SEB använder sig utav CefSharp som är en inbyggd webbläsare till C#-program. Client och Browser kommunicerar direkt med varandra för att kunna övervaka händelser som sker i webbläsaren.

## 3.2 Safe Exam Browsers loggar

SEB loggar relevant och detaljerad information om uppstartningsprocessen samt händelser medan applikationen kör. Detta gör att man lätt kan detektera var eventuella fel har uppstått vilket underlättar felsökning för utvecklare. Det finns en loggfil för varje SEB-process samt en loggfil för webbläsaren. För varje händelse som loggas sparas tidpunkt, loggkategori samt information om vad som har hänt. De fyra loggkategorierna som finns är: Error, Warning, Info, Debug.

## 3.3 Safe Exam Browsers konfiguration

För att kunna ändra inställningar som förändrar applikationens beteende finns ett konfigurationsverktyg till SEB. Verktyget har ett grafiskt användargränssnitt som gör det möjligt att ändra inställningarna. Verktyget ger även möjlighet att kryptera konfigurationsfilerna så att de inte kan ändras av provtagare.

## 3.4 Safe Exam Browsers detektion av virtuella maskiner

SEB försöker detektera virtuella maskiner för att provtagare inte ska kunna starta applikationen i en virtuell miljö. Detta eftersom provtagaren då kan öppna sina anteckningar på den fysiska datorn och sedan köra SEB i en virtuell miljö för att kringgå lockdown-funktionen.

Den nuvarande detektionen av virtuella maskiner som finns i SEB letar huvudsakligen i systeminformation och registervärden efter strängar som förekommer i virtuella maskiner. SEB tittar även på unika hårdvaru-ID:s samt BIOS-information för att titta om det finns suspekta strängar.

Vid uppstart av applikationen genomförs totalt fem olika kontroller:

| | |
|---|---|
| **1. BIOS** | Applikationen letar efter specifika strängar i BIOS-informationen som i virtuella maskiner ofta innehåller VM-tillverkarens namn. |
| **2. Registervärden** | Applikationen tittar på olika registervärden i Windows för att säkerställa att inga bannlysta strängar finns i registret. |
| **3. MAC-adress** | Applikationen tittar om MAC-adressen är suspekt. |

**4. Processornamn** Applikationen letar efter suspekta strängar i processorns namn.

**5. Plug-n-Play** Applikationen kontroller om virtuella Plug-n-Play-enheter finns vilket är vanligt bland virtuella maskiner.

## 3.5 Trusted Platform Module

Trusted Platform Module (TPM) är ett kryptografiskt säkerhetschip som finns i diverse datorer och inbyggda system [6]. I Windows används TPM-enheten bland annat till att skydda data och att säkerställa att känslig mjukvara i datorn inte är modifierad när operativsystemet startar. Windows 11 är det första Windows-operativsystemet där en TPM-enhet är ett systemkrav [7]. En TPM-enhet innehåller en så kallad Endorsement Key som består av både en publik och privat nyckel.

## 3.6 Timing Attack

Timing Attack innebär att man mäter tidsvariationer i systemets beteende under vissa specifika operationer. Detta kan användas för att detektera en virtuell maskin eftersom den behöver släppa kontrollen till den fysiska datorn för att utföra vissa operationer och detta tar längre tid. För att göra detta kan man mäta tidsvariationer i hur datorn till exempel läser minne, skriver till lagringsenheter och uppför sig vid mycket belastning. Att detektera virtuella maskiner genom Timing Attacks är utmanande och kräver noggranna analyser av hur virtuella maskiner uppträder. Dessa metoder är inte alltid pålitliga då de kan ge positiva resultat även fast applikationen inte körs i en virtuell maskin.

## 3.7 Mozilla Public License 2.0

Mozilla Public License 2.0 (MPL 2.0) [2] är en öppen källkodslicens som används för att licensiera mjukvaruprojekt. Den tillåter användare av licensen att använda, modifiera och distribuera koden. Detta gäller även kommersiella mjukvaruprojekt under förutsättning att de ändringar som görs också släpps under MPL 2.0-licensen. Eventuella ändringar eller tillägg till den ursprungliga koden måste publiceras under samma licens. Nya filer som inte går under licensen MPL 2.0 behöver ej publiceras [8].

# 4. Genomförande

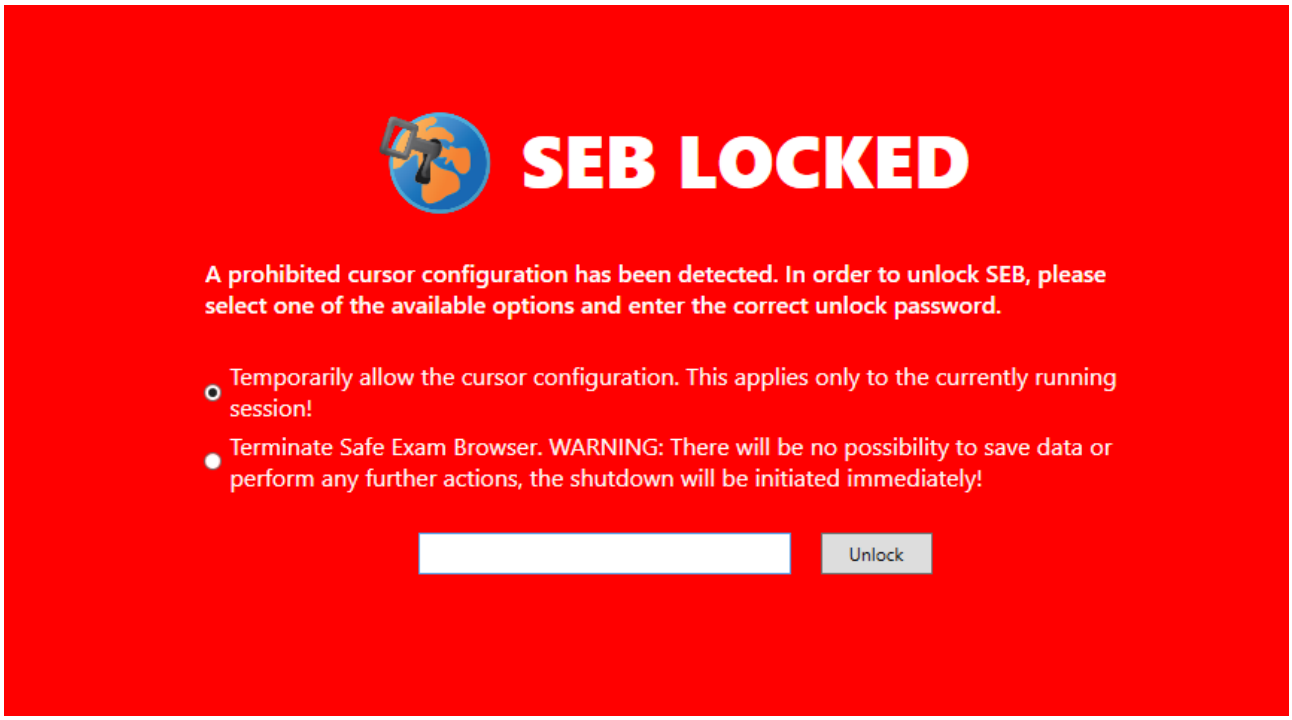
Avsnitten i detta kapitel följer en gemensam struktur. Först presenteras relevant fakta och problemställningen. Utöver detta förklaras även problematiken som lett företaget till att vilja förändra funktionaliteten. Sedan redogörs lösningen som implementerats för att uppnå målet.

För att säkerställa att man under projektets gång inte bryter mot MPL 2.0 satte sig projektmedlemmarna in i villkoren för licensen för att konstatera vad denna licens ställer för krav på utvecklare som modifierar SEB. Handledaren på företaget hade en förhoppning om att det ska gå att distribuera den egna versionen av SEB utan att behöva publicera nya filer. Detta ger möjlighet att dölja kod som implementerar säkerhetsfunktioner vilket gör det svårare att kringgå dessa.

Efterforskningarna som gjorts är inte uttömmande, men visade att MPL 2.0 är en så kallad filbaserad copyleft-licens. Det innebär att man vid distribution behöver publicera de filer som går under MPL 2.0, men att man inte behöver publicera nya filer så länge dessa inte släpps under MPL 2.0. Därför beslutades det i samråd med företaget att i möjligaste mån placera kod som skrivs i nya filer.

## 4.1 Delmål 1 - Den röda låsskärmen

Det som avsågs förändras på denna punkt var den röda låsskärmen. Den röda låsskärmen låser datorn och dyker upp när något suspekt händer. Provtagaren kan då inte fortsätta att skriva provet och kan inte heller använda datorn till annat. Händelser som utlöser låsskärmen kan till exempel vara när förändringar i vissa specifika registervärden sker, eller om provtagaren startar om datorn och återstartar provet.




**Figur 1.** Den röda låsskärmen i SEB

Den som konstruerar provet väljer om denne vill konfigurera ett lösenord till låsskärmen. Om man gör detta krävs det att man skriver in lösenordet för att låsa upp datorn efter att låsskärmen utlösts. När ett lösenord till låsskärmen inte är konfigurerat så kan provtagaren själv stänga ner rutan, och provvakten får inte heller någon notis att det har hänt.

Om låsskärmen utlöses är det fördelaktigt att använda både serverapplikationen och verifieringverktyget. Serverapplikationen ser till att provvakten får en notis när låsskärmen utlöses på en provtagares dator. När man får denna notis är tanken att man ska gå till datorn i fråga, låsa upp den och sedan verifiera programkodens integritet med hjälp av verifieringsverktyget.

Serverapplikationen eller verifieringsverktyget används normalt sett inte av företaget, och SEB tillåter inte att inaktivera låsskärmen. Det förekommer att låsskärmen dyker upp under prov och detta beror inte alltid på att faktiskt fusk har förekommit. I detta läge rekommenderar företaget sina kunder att använda verifieringsverktyget till att verifiera applikationens integritet.

Låsskärmen orsakar i nuläget oftast förvirring och problem för företagets kunder. Eftersom de tillhörande verktygen inte används erbjuder låsskärmen inte någon hjälpsam funktionalitet för företagets kunder. Därmed uppstod önskemålet om att stänga av låsskärmen och i stället skulle information om varför låsskärmen utlöstes skickas till företagets servrar. Företaget kan då också om de önskar implementera en egen låsskärm i webbapplikationen som kan stängas automatiskt om läraren godkänner att eleven får fortsätta.

Det ansågs att det enklaste och mest modulära sättet att uppnå detta var att skicka informationen till den inbyggda webbläsaren. Eftersom man alltid är (eller åtminstone ska vara) inne på företagets webbplats innebär detta att informationen kan nå deras servrar med denna lösning. Detta uppnås på ett enkelt sätt genom att injicera JavaScript i webbläsaren. Stöd för att göra detta fanns redan i kodbasen, vilket gjorde detta till en lösning som var lätt att implementera.

Inledningsvis löstes delmålet genom att ändra i metoden som visar låsskärmen på användarens skärm. I början av denna metod skickades ett meddelande att låsskärmen har utlösts direkt till webbläsaren. När man sedan löste delmål 3 insåg man att det räcker att logga händelsen och sedan inte tillåta låsskärmen att visas. Den slutliga lösningen blev alltså väldigt simpel. Man loggar att låsskärmen har utlösts, och implementationen i delmål 3 ser sedan till att logghändelsen kommuniceras till webbläsaren med hjälp av JavaScript.

## 4.2 Delmål 2 - Omkonfigurering via webbserver

Problemställningen uppstod på grund av att SEB:s standardinställningar låser datorn så fort applikationen startar. Detta gör det svårt för provtagaren att på ett enkelt sätt skriva in engångsnyckeln för att komma in på rätt prov. Detta eftersom man inte kan byta till ett annat program, och dessutom inte tillåts använda klipp-och-klistra-funktionen i Windows. Det har även upplevts som störande av användarna att inte kunna göra annat på datorn innan man faktiskt har påbörjat provet.

När applikationen startar försöker den att ladda in inställningar från en lokalt lagrad konfigurationsfil. Denna standardfil finns alltid så länge man inte raderat den från sitt filsystem. I det fallet att filen inte finns laddas inställningar in från kod. De hårdkodade inställningarna ser till att applikationen kan starta och att du kommer till en informationssida på SEB:s webbplats. Webbplatsen ber dig skapa en konfigurationsfil och när en av företagets kunder kommer in på den uppstår ofta förvirring.

För att omkonfigurera klienten vid behov har SEB funktionalitet som stödjer detta. Detta sker genom att ladda ned en konfigurationsfil med filändelsen .SEB i den inbyggda webbläsaren. När man gör detta kommer klienten identifiera att en .SEB-fil har laddats ner och en metod för omkonfiguration anropas. Applikationen startar sedan om och laddar samtidigt in den nya konfigurationsfilen. I samband med detta skrivs den lokala konfigurationsfilen över med de nya inställningarna.

Lösningen som valdes var då att stänga av funktionaliteten som försöker ladda in den lokala konfigurationsfilen vid uppstart. Istället för att ladda in den lokala konfigurationsfilen sätts inställningarna genom hårdkodade värden. De hårdkodade värdena har ändrats och anpassats efter företagets behov.

Innan de hårdkodade värdena sätts görs ett API-anrop till företagets webbplats för att få en länk till en nedladdningsbar .SEB-fil. Denna länk sätts sedan som landningssida i den inbyggda webbläsaren. Detta leder till att applikationen kommer att gå in och ladda ner filen i webbläsaren, vilket i sin tur leder till att en omkonfigurering sker med inställningarna i den nedladdade .SEB-filen. Detta medför att applikationen startar om, och för att inte förvirra användaren döljs alla fönster vid den initiala uppstarten.

Detta ansågs vara en fördelaktig lösning då det möjliggör att man dynamiskt kan ändra vilken fil man vill att klienten ska ladda ner och därmed enkelt ändra standardinställningarna i samtliga kunders applikationer. Det kräver dock en internetuppkoppling, men SEB är konstruerat på ett sådant sätt att den inte startar om internetuppkopplingen saknas så detta ansågs inte vara ett problem.

## 4.3 Delmål 3 - Skicka logghändelser till servern

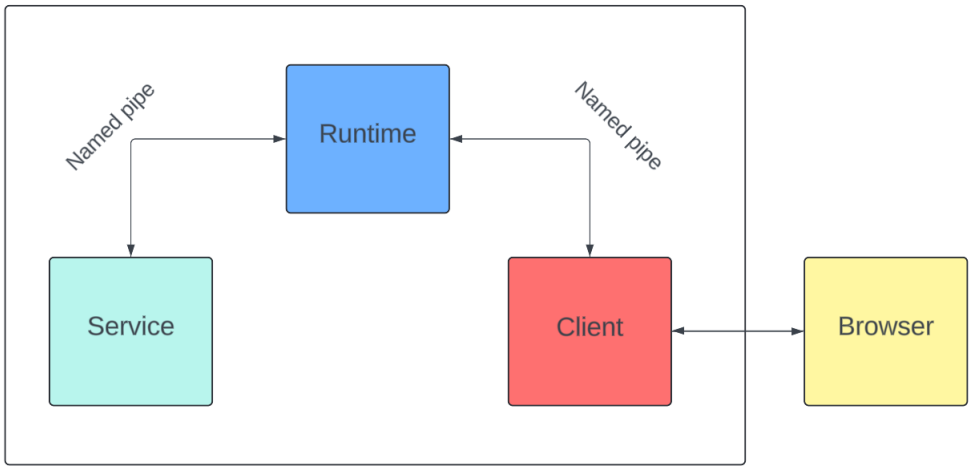
I SEB loggas en stor mängd händelser i textfiler som lagras lokalt på användarens dator. Varje process (Runtime, Service, Client & Browser) har en egen loggfil. Det finns fyra olika loggkategorier. Vilken detaljnivå på loggkategorin som används bestäms i konfigurationsfilen, och endast händelser av den valda detaljnivån och högre sparas i loggfilerna.

I vissa fall är det intressant för företaget att ta del av loggfilerna. Då kan de till exempel användas för att felsöka och åtgärda saker såsom körningsfel. Eftersom loggfilerna sparas lokalt på provtagarens dator, behöver någon leta upp och skicka loggfilerna till företaget när de har blivit ombedda att göra detta. Detta upplevs ofta som krångligt och kräver en del teknisk kunskap som inte alla besitter. Det var på grund av denna problematik som önskemålet om att få logghändelser direkt till företagets servrar uppstod.

I samråd med handledaren på företaget beslutades att åstadkomma detta med hjälp av JavaScript-injicering i den inbyggda webbläsaren. Den kod som injiceras i webbläsaren anropar en JavaScript-metod som företaget har skapat för detta ändamål. Företaget ordnar också själva så att de meddelanden som kommer till webbläsaren tas emot och hanteras på det sätt de önskar.

En del av företagets önskemål var att logghändelser inte längre sparas till filer under körning, och endast skickas till deras servrar. Detta ledde till en lösning där man stänger av den del av koden som sparar logghändelser till filer på datorn, och istället ordnar så att alla logghändelser kommuniceras genom JavaScript-injicering i webbläsaren.

Funktionalitet för att injicera JavaScript i webbläsaren finns redan i SEB. Detta sker genom ett metodanrop i Client-processen. Det innebär att informationen som ska kommuniceras med hjälp av JavaScript behöver finnas tillgänglig för Client-processen. För att processerna ska kunna skicka information mellan varandra använder de sig av Named pipes. Dock har inte alla processer pipes sinsemellan, utan det är Runtime som har en pipe vardera till Service och Client. Det finns ingen pipe mellan Service och Client. För en grafisk illustration av kommunikationen mellan processerna, se figur 2.




**Figur 2.** Överblick av kommunikationen mellan processerna i SEB

Implementationen där loggfilerna raderas visade sig dock vara relativt komplex och kräva mycket ändringar i kodbasen. Eftersom Service och Client saknar direkt kommunikation sinsemellan krävdes det ett sätt att förmedla logghändelser från Service till Client. De två uppenbara sätt att ordna detta var att antingen skapa en pipe mellan processerna, eller att låta Service skicka logghändelser till Runtime som i sin tur vidarebefordrar dessa till Client. Utöver detta behövde man även skapa nya meddelandetyper i det befintliga kommunikationsprotokollet. Implementationen av detta påbörjades men ett tag in i utvecklingen insåg man att det blir onödigt komplicerat.

Man utvärderade lösningen på nytt med handledaren på företaget och kom istället fram till att behålla loggfilerna och låta Client-processen periodvis läsa av innehållet i loggfilerna. Sedan vidarebefordrar man detta till webbläsaren genom JavaScript-injicering. Denna lösning ger högre kohesion och lägre koppling i koden, vilket leder till att implementeringen blev avsevärt enklare och att kodbasen blir lättare att underhålla.

## 4.4 Delmål 4 - Möjliggör manuell uppstart av SEB

Med företagets nuvarande lösning måste provtagaren starta applikationen genom att först gå in på företagets webbplats via sin webbläsare och sedan ange en engångsnyckel. När en korrekt engångsnyckel angivits initierar webbplatsen uppstart av applikationen med en specifik konfiguration. Detta har varit det enda sättet för företaget att säkerställa att applikationen laddas med rätt konfiguration. Man ville möjliggöra att starta applikationen direkt på datorn genom att öppna SEB och låta provtagaren skriva in engångsnyckeln i den interna webbläsaren. Detta förbättrar användarvänligheten genom att erbjuda två olika sätt att starta applikationen.

För att möjliggöra detta behövde man först och främst säkerställa att SEB alltid laddar in företagets landningssida vid uppstart. Den lösning som föreslogs var att man modifierar SEB till att ladda in konfigurationsfilen genom ett API-anrop vid uppstart.

Detta delmål löstes samtidigt som delmål två då problemställningarna var nära besläktade med varandra. En mer ingående beskrivning på hur detta gick till återfinns i avsnitt 4.2.

## 4.5 Delmål 5 - Säkerhetsfunktioner

### 4.5.1 Detektion av virtuella maskiner

Det finns ett antal olika metoder i SEB för att detektera att applikationen körs i en virtuell maskin. Dessa ger grundläggande skydd mot attackerarmodellen men det finns publika lösningar för att kringgå skyddet [9]. För en mer ingående beskrivning av dessa skyddsmetoder, se avsnitt 3.4. Dessa metoder fångar upp många fall där den virtuella maskinen är omodifierad. Är man motiverad och kunnig kan man dock ta sig runt dessa genom att ändra systeminformationen som den virtuella maskinen visar. Det finns även publika mjukvaruprojekt vars syfte är att undvika VM-detektion. Exempel på dessa är [10] och [9].

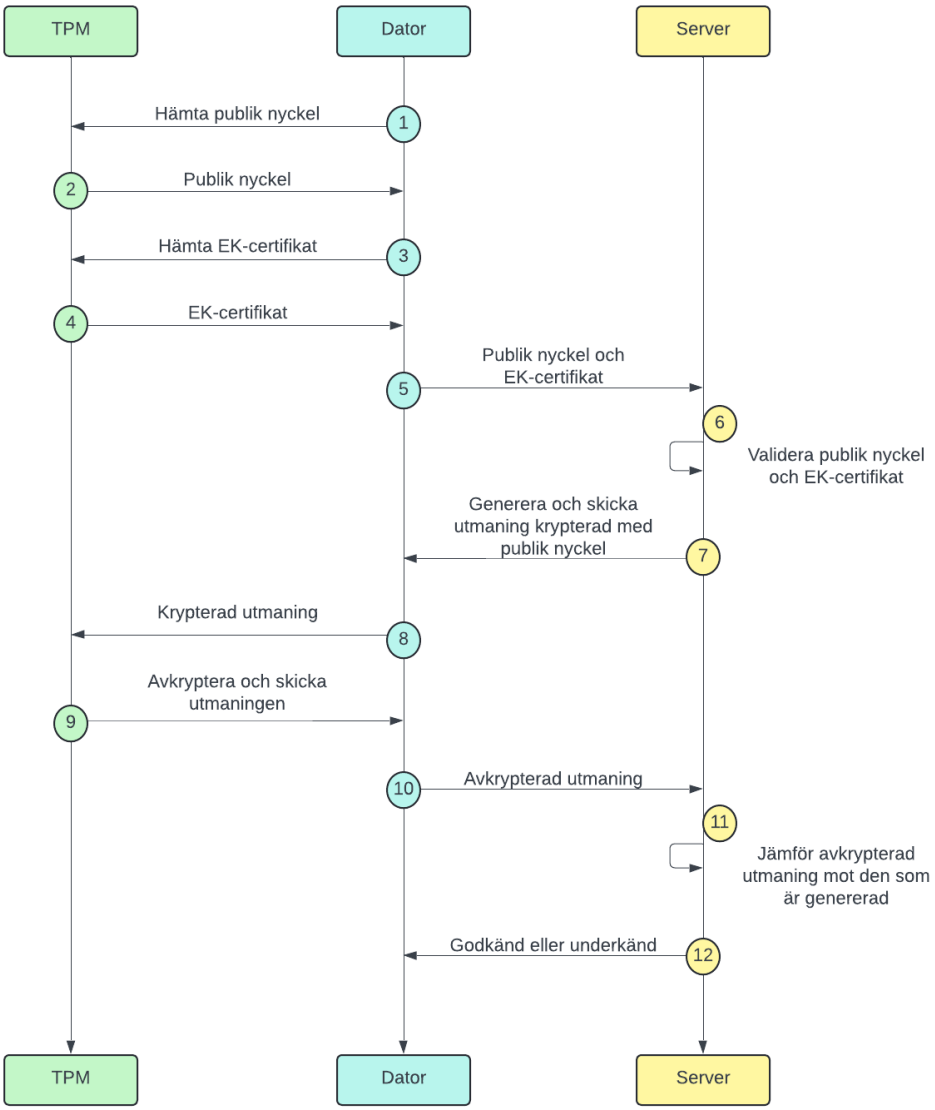
En svaghet i SEB:s VM-detektion är det faktum att applikationen har öppen källkod. Det ger vem som helst möjlighet att läsa källkoden för att lista ut hur man ska kringgå säkerhetsfunktionerna. Det var svagheterna i SEB:s nuvarande VM-detektion som ledde till ett önskemål från företaget om att förstärka VM-detektionen i applikationen.

#### 4.5.1.1 Trusted Platform Module

Då grundläggande skydd mot virtuella maskiner redan existerar började projektmedlemmarna titta på lite mer sofistikerade sätt att detektera en VM. En idé som fanns var att använda TPM-enheten. Detta är en kryptografisk enhet som förekommer på moderna datorer. Windows stödjer TPM i Windows Vista [11] och framåt. I Windows 11 är TPM ett systemkrav [7]. Detta innebär att man kan kräva att en provtagare som använder Windows 11 ska ha en legitim TPM-enhet.

I TPM-enheten finns det en kryptografisk nyckel kallad Endorsement Key (EK). På fysiska TPM-enheter är nyckeln inbränd i hårdvaran och går inte att ändra. Den privata delen av EK går inte att komma åt från operativsystemet. Tillverkaren av TPM-enheten har ett certifikat som alla deras EK:s är signerade med.

Den första idéen var en metod som projektmedlemmarna kommit på själva, och detta var att använda sig av TPM-enheten för VM detektion. Tanken var att säkerställa att en dator har en legitim TPM-enhet. Detta görs genom attestering av EK där man bevisar att enheten har både den privata och den publika nyckeln. Därför måste man ha en tredjeparts-server som hjälper till att validera det certifikat och den publika nyckel som skickas. Servern genererar även en utmaning som TPM-enheten måste klara av för att bli godkänd. Utmaningen genomförs genom att servern krypterar ett meddelande med den publika nyckeln som TPM-enheten behöver avkryptera med den privata nyckeln och skicka tillbaka till servern för att klara av utmaningen. Detta gör att servern kan garantera att TPM-enheten även har den privata nyckeln och på detta sätt kan man säkerställa att TPM-enheten är legitim. Figur 3 visar hur TPM- attestering skulle kunna gå till i SEB.




**Figur 3.** Överblick av hur attesteringsprocessen skulle kunna gå till i SEB

Det finns många sätt att implementera attestering. Figur 3 är endast ett förslag på hur man skulle kunna göra detta. Vad gäller metoden i figur 3 är det framför allt det första och sista steget som kräver noggrann beaktning på hur det ska fungera. Ska klienten eller servern initiera händelseförloppet? Var ska servern rapportera resultatet av utmaningen? Att låta servern hantera så mycket som möjligt ger bättre säkerhet i applikationen.

Eftersom utveckling av en attesteringsfunktion visade sig vara för tidskrävande lämnas designen och implementationen av detta till företaget. I stället valde man att implementera en funktion som extraherar och kontrollerar information från TPM-enheten utan att attestera. För att få ut den data som behövdes användes Windows-biblioteket Ncrypt som tillhandahåller ett gränssnitt till TPM-enheten. Genom Ncrypt-biblioteket extraherades *Manufacturer ID* från TPM-enheten. Detta informationsfält innehåller data som unikt identifierar tillverkaren av TPM-enheten. Man jämförde sedan *Manufacturer ID* mot en lista med alla legitima TPM-tillverkare som tillhandahålls av Trusted Computing Group [12]. Om informationen från TPM-enheten inte matchar en tillverkare kan man med någorlunda säkerhet konstatera att TPM-enheten inte är äkta. För att kunna göra dessa kontroller måste man även se till att datorn och operativsystemet stöder TPM.

Lösningen testades på följande virtuella maskiner: VMWare, VirtualBox och QEMU. Pålitligheten i både den implementerade funktionen och TPM-attestering diskuteras mer ingående i avsnitt 6.2.1.

#### 4.5.1.2 Grafikprocessor

En märklig sak som upptäcktes på virtuella maskiner var att grafikminnesanvändningen för processer som utför grafiska operationer är noll. Detta betyder att processen inte använder något grafikminne för att utföra grafiska operationer, vilket förmodligen inte är möjligt på en fysisk enhet med ett fungerande grafikkort. För att utforska möjligheten att detektera virtuella maskiner på detta sätt testades flera olika virtuella miljöer under projektets gång. De som testades var: VMware, VirtualBox, QEMU och Hyper-V.

Tillvägagångssättet som valdes för att detektera detta beteende var att använda sig av "Performance Counters" (Prestandaräknare), vilket är ett sätt att övervaka resursanvändning i en dator. Det beslutades att på detta sätt övervaka processen dwm.exe. DWM står för Desktop Window Manager och processen ansvarar för att sätta samman och rendera fönster i Windows. Processen används sedan Windows Vista, och är systemkritisk för att Windows 10 och 11 ska fungera [13], [14]. Att den är systemkritisk innebär att systemet inte fungerar utan den. Den kan inte stängas av, och kommer automatiskt att återstartas om man lyckas stänga ner den eller om den kraschar.

Det beslutades att använda Windows-biblioteket "pdh.dll" för att extrahera information om dwm.exe. PDH står för Performance Data Helper och biblioteket ger åtkomst till en stor mängd prestandaräknare som Windows bokför. Den räknare som används är "GPU Process Memory" som rapporterar hur mycket grafikminne som tas i anspråk av en process. Man kan extrahera denna information i Windows PowerShell med följande två kommandon:
*"Get-Counter -Counter "\GPU Process Memory(pid_x*)\Local Usage"*
*"Get-Counter -Counter "\GPU Process Memory(pid_x*)\Non Local Usage"*
där x byts ut mot det process-ID:t som tillhör den process man vill observera.

För att säkerställa att inget grafikminne tas i anspråk av en process behöver man titta på både "Local Usage" och "Non Local Usage". Skillnaden på dessa två är var i datorn minnet är allokerat. "Local Usage" är minne som allokerats lokalt på grafikkortet, det vill säga i VRAM. "Non Local Usage" är minne som lånats av andra komponenter i datorn, i de allra flesta fall RAM-minne från processorn.

I stora drag fungerar metoden som extraherar informationen om dwm.exe:s grafikminnesanvändning på följande sätt:

1. Hämta dwm.exe:s Process ID.
2. Skicka en förfrågan för att hämta data från de relevanta prestandaräknarna.
3. Addera de två datapunkterna Local och Non Local.
4. Kontrollera om summan av de två är noll.

Denna information kan sannolikt inte på egen hand användas för att avgöra huruvida SEB körs i en virtuell maskin eller inte. Funktionen rapporterar endast resultatet och är tänkt att fungera som en indikation på fusk. Varför dessa prestandaräknare är noll är inte fastställt och teorier kring detta diskuteras i avsnitt 6.2.2.

#### 4.5.1.3 Timing Attacks

Timing Attacks är en vanligt förekommande metod för att detektera virtuella maskiner [15]. Därför valde man att under projektets gång undersöka möjligheten att implementera sådan funktionalitet i applikationen. Det visade sig dock vara svårare än väntat då dagens virtuella maskiner har bra prestanda och inte avviker mycket från en fysisk dator i hur lång tid olika operationer tar.

För att undersöka huruvida Timing Attacks var något som kunde användas till att detektera en virtuell maskin testades två publika lösningar. Dessa var Pafish [16] och Al-Khaser [17]. De testades grundligt på två olika fysiska datorer och virtuella maskiner som kördes i Hyper-V och VMware. Resultaten visade sig vara bristfälliga, se avsnitt 5.4.3 för detaljer.

### 4.5.2 Modified Client

SEB använder Windows-certifikat för att säkerställa att filen som körs är signerad med ETH Zürichs certifikat. Detta går att kringgå lätt eftersom källkoden till applikationen är publik. Exempelvis kan detta göras genom att ta bort den del av koden där applikationen utför denna kontroll. Detta tillvägagångssätt kan användas för att köra en modifierad version av SEB vid ett provtillfälle, och ger då inget bra skydd mot en sofistikerad angripare. Dock kommer delar av koden som producerats i detta projekt att vara privat, vilket gör att företaget kan detektera att viss funktionalitet saknas, såsom loggmeddelanden i avsnitt 4.3. Med detta i åtanke beslutades att skyddet mot Modified Client i nuläget inte behöver förstärkas.

## 4.6 Testning

Under projektets gång genomfördes en stor mängd förändringar i kodbasen. Viss funktionalitet inaktiverades, annan funktionalitet modifierades och det implementerades även helt ny funktionalitet. Dessa förändringar orsakade att SEB:s existerande enhetstester slutade att fungera som de skulle. För att säkerställa funktionaliteten i den nya koden behövde man även implementera nya enhetstester.

Testningen sparades till sist då man inte ville behöva gå in och ändra flera gånger i testkoden under utvecklingens gång. Relativt små förändringar krävdes för att låta de ursprungliga testerna fungera normalt. Den funktionalitet som inaktiverats behölls i kodbasen för att tillåta återaktivering av funktionaliteten under enhetstesterna. Detta medför att den inaktiverade funktionaliteten fortsätter att testas vilket gör det enklare för företaget att återställa funktionaliteten om så önskas.

# 5. Resultat

De delmål som sattes upp för projektet har uppnåtts. Delmål 1–4 hade alla tydliga avgränsningar i sina problemställningar, medan delmål 5 hade en mer öppen problemställning. Detta har lett till att delmål 5 omfattar mer efterforskning och spekulativa slutsatser.

Samtliga delmål har implementerats på ett sådant sätt att det uppfyller företagets krav om att inte bryta mot MPL 2.0 samt underlätta synkronisering av företagets kodbas med SEB:s kodbas. Efterforskningarna visar att det går att distribuera en modifierad version av SEB utan att behöva publicera kod som placerats i nya filer.

## 5.1 Delmål 1 - Den röda låsskärmen

Inaktiveringen av den röda låsskärmen fungerar väl och lösningens simplicitet bör leda till att inga oväntade problem uppstår i applikationen. Lösningen är kort och består endast av fyra rader kod vilket bibehåller låg koppling och hög kohesion i kodbasen. Detta var möjligt eftersom implementationen i delmål 3 ordnade så att loggmeddelanden skickas till företagets servrar. Under prov kommer den röda låsskärmen inte att utlösas under några omständigheter. Dock återaktiveras den under körning av enhetstester.

## 5.2 Delmål 2 & 4 - Omkonfigurering via webbserver & möjliggör manuell uppstart av SEB

Dessa delmål fick en gemensam lösning eftersom båda två kräver att applikationen startar med vissa specifika inställningar. Det var då enklast att lösa båda problemen samtidigt. Lösningen medför att applikationen tar längre tid att starta, eftersom en omkonfigurering sker direkt efter att applikationen initialt har startat. Tiden det tog för applikationen att starta var ungefär tolv sekunder, vilket är en ökning på fem sekunder jämfört med omodifierad version. Vid den initiala uppstarten, det vill säga innan omkonfigurationen, har samtliga fönster dolts. Detta för att inte skapa förvirring eller lura användaren att uppstarten är klar.

Det finns förbättringar att göra för att snabba upp denna funktionalitet. Rekommendationer kring hur detta kan genomföras lämnas i avsnitt 6.1.

## 5.3 Delmål 3 - Skicka logghändelser till servern

Delmål tre löstes genom att en gång i sekunden läsa av de nuvarande loggfilerna och skicka de raderna av loggen som inte redan har skickats till webbläsaren. Detta sker för loggfilerna tillhörande *Client*, *Runtime*, *Service* och *Browser*. Funktionaliteten har implementerats i en egen fil, vilket har medfört att endast små förändringar i SEB:s kodbas har krävts. Detta gör framtida synkronisering av företagets kodbas med SEB:s kodbas avsevärt enklare.

## 5.4 Delmål 5 - Säkerhetsfunktioner

### 5.4.1 Trusted Platform Module

Att detektera en virtuell maskin genom att extrahera information från TPM-enheten fungerar bra utan falska positiva resultat i de tester som utförts. Trusted Computing Group tillhandahåller en lista med *Manufacturer ID*, vilket är pålitliga företag som utvecklar TPM-enheter. Listan används för att kontrollera om TPM-enheten kommer från en pålitlig tillverkare eller inte. De virtuella maskiner som testats har alla virtualiserade eller emulerade TPM-enheter, och anger oftast sitt eget företagsnamn som *Manufacturer ID* i TPM-enheten. Ett undantag till detta som funnits är QEMU och VirtualBox som kringgår den implementerade kontrollen eftersom *Manufacturer ID*:t som rapporteras är "IBM ", som finns med på listan över pålitliga tillverkare.

En mer robust metod att detektera virtuella maskiner med hjälp av TPM-attestering upptäcktes. Detta går att läsa mer om i avsnitt 4.5.1 och 6.2.1.

Funktionen ger en god förbättring av skyddet mot attackerarmodellen eftersom man inte lyckats finna publika lösningar som modifierar TPM-enhetens information. Det krävs då att attackeraren har tillräcklig kunskap för att modifiera programvaran för att kringgå detta. Om man implementerar TPM-attestering kommer detta att erbjuda ett bättre skydd.

### 5.4.2 Grafikprocessor

Den önskade funktionaliteten uppnåddes och applikationen extraherar information om grafikminnesanvändning från prestandaräknarna i Windows. Lösningen säkerställer även att ett körningsfel inte orsakar problem. Om detta sker loggas detta till företagets servrar.

Metoden ger endast en indikation på att en virtuell maskin används och låser alltså inte ute provtagaren från att fortsätta provet. Lösningen ger möjlighet till företagets personal att använda informationen som beslutsunderlag i ett fall där man manuellt behöver avgöra om en provtagare använder en virtuell maskin eller inte. För att avgöra huruvida funktionaliteten går att använda till fuskdetektion bör företaget samla in data och analysera den.

Varför den grafiska minnesanvändningen är noll i prestandaräknarna på virtuella maskiner är inte helt fastställt. Diskussion kring detta finns i avsnitt 6.2.2. Eftersom det finns oklarhet kring orsaken är det svårt att utvärdera huruvida funktionaliteten kommer att ge ett bättre skydd mot attackeraren. Förhoppningen är att det åtminstone ska krävas GPU-passthrough för att kringgå detta. Om så är fallet är det en god förbättring av säkerheten i applikationen.

### 5.4.3 Timing Attacks

De testerna som utfördes på både de virtuella och fysiska datorerna gav dåliga resultat. Det största problemet är att de publika mjukvarorna som testades gav falska positiva utfall på fysiska datorer. Varken Al-Khaser eller Pafish erbjöd starka metoder för att detektera en virtuell maskin utan att även orsaka falska positiva utfall.

Testresultaten ledde att man nedprioriterade en Timing Attack-implementation under projektets löptid. I slutändan fanns det inte tid till detta. Endast grundläggande efterforskning har gjorts och den insamlade informationen lämnas som en utgångspunkt till företaget om de väljer att utforska Timing Attacks.

## 5.5 Testning

Funktionaliteten till de enhetstester som finns i originalapplikationen återställdes relativt enkelt. Trots att viss funktionalitet inaktiverats i applikationen återaktiverar enhetstesterna funktionaliteten och testar på samma sätt som tidigare. Detta möjliggörs av genomtänkta lösningar i samtliga delmål där man minimerat förändringar i originalkoden. Lösningen valdes för att kunna bevara testningen av originalfunktionaliteten ifall företaget skulle vilja återinföra dessa i framtiden.

Nya enhetstester har skapats till all ny funktionalitet. Dessa använder samma testmetodik som de enhetstester som redan fanns i applikationen. Den koden som skapats under projektet passerar de nya enhetstesterna.

# 6. Diskussion

Nedan diskuteras endast delmål 2, 4 och 5 då genomförandet av delmål 1 och 3 inte anses resulterat i något intressant diskussionsunderlag.

## 6.1 Delmål 2 & 4 - Omkonfigurering via webbserver & möjliggör manuell uppstart av SEB

Den implementerade lösningen gav det önskade beteendet men försämrar prestandan i applikationen, vilket försämrar användarupplevelsen. Att genomföra en omkonfiguration förlänger applikationens starttid eftersom en fil laddas ner och SEB måste starta om. En alternativ lösning som åtgärdar problemet har identifierats. Istället för att ladda ned en konfigurationsfil kan man göra ett API-anrop som hämtar konfigurationsdatan. Man kan då sätta inställningarna direkt, och låta SEB starta med den önskade konfigurationen vid första uppstarten. Detta bör minska uppstartstiden med cirka 40 %. Denna lösning medför inga nackdelar och bibehåller fördelarna med den tidigare lösningen. Lösningen åtgärdar även den största nackdelen, vilket var prestandaförsämringen.

## 6.2 Delmål 5 - Säkerhetsfunktioner

### 6.2.1 - Trusted Platform Module

Som med de flesta säkerhetsfunktioner finns det sätt att kringgå TPM-kontrollen som implementerats. Som presenterades i avsnitt 5.4.1 ger lösningen inget skydd mot virtuella maskiner som körs i QEMU eller VirtualBox. Detta beror på att det *Manufacturer ID* som syns i maskinen är “IBM “. Eftersom IBM finns i listan med pålitliga TPM-tillverkare som Trusted Computing Group tillhandahåller detekterar den nya funktionaliteten inte dessa virtuella maskiner. En person med goda programmeringskunskaper skulle även kunna ändra det *Manufacturer ID* som visas i den virtuella maskinen och på detta sätt kringgå lösningen.

En mer robust användning av TPM-enheten är genom attestering. Genom attestering kan man säkerställa att TPM-enheten är från en pålitlig tillverkare. Ett exempel på hur attestering kan gå till finns i figur 3. Virtualiserade TPM-enheter kommer att upptäckas vid attestering eftersom de saknar certifikat från en pålitlig tillverkare. Att kringgå attestering är svårt men inte omöjligt. Det är osannolikt att en person som försöker knäcka SEB skulle hitta ett sätt att kringgå attestering då det kräver djup kunskap.

Ett sätt att undvika detektion av en virtuell maskin vid attestering är genom TPM-passthrough. TPM-passthrough tillåter den virtuella maskinen att använda datorns TPM-enhet, och man slipper då att virtualisera TPM-enheten. Den virtuella maskinen använder då datorns riktiga TPM-enhet. Den virtuella maskinen och datorn kommer att dela på TPM resurser vilket kan leda till konflikter som orsakar problem [18]. TPM-passtrough finns exempelvis i QEMU, men kräver manuell konfiguration.

### 6.2.2 - Grafikprocessor

Varför beteendet med att prestandaräknaren är noll förekommer är inte helt fastställt. En teori om vad detta beror på är att de virtuella maskinerna som testats använder gamla drivrutiner. De använder en gammal version av Windows Display Driver Model (WDDM) som inte stödjer de prestandaräknare som kontrolleras. Samma sak händer i Aktivitetshanteraren som kräver WDDM 2.0 eller högre för att visa GPU-information korrekt [19]. Alla de virtuella maskiner som undersökts i Windows 11 har använt sig av WDDM 1.3, vilket är märkligt eftersom WDDM 2.0 är ett systemkrav i Windows 11 [7].

Ett sätt att uppdatera en virtuell maskin till WDDM 2.0 eller högre är att ordna GPU-passthrough. Då används samma drivrutiner som finns på den fysiska datorn. Att GPU-passthrough skulle lösa problemet med att prestandaräknaren rapporterar noll är inte testat. Därför kan det enbart ses som en teori. Om man ordnar GPU-passthrough är det även svårt att veta om det är WDDM-versionen eller något annat som i så fall får prestandaräknaren att fungera. GPU-passthrough kräver oftast manuell konfiguration, vilket gör att det kan vara svårt att ordna för någon utan teknisk kunskap.

Om prestandaräknarens beteende beror på WDDM-versionen kan lösningen som implementerats sluta fungera korrekt om VM-tillverkaren uppdaterar grafikdrivrutinen till att använda en nyare WDDM-version.

### 6.2.3 - Kapplöpningen i datasäkerhet

Att säkerställa att en digital provmiljö inte körs i en virtuell maskin är svårt. I synnerhet när det kommer till prov där provtagare tar med sin egen dator eftersom man då inte kan säkra vare sig mjuk- eller hårdvaran i datorn. Det blir därför en kapplöpning där utvecklare av provprogramvara ständigt stänger säkerhetsluckor, och attackerare letar efter nya. Det bästa skyddet mot användande av virtuella maskiner eller fusk i allmänhet vid digital provtagning är att låta provtagarna skriva provet på datorer som de inte har tillgång till utanför provtillfället. Ett exempel på en sådan miljö kan vara en datorsal i en skola, där skolans IT-administratörer har full kontroll över all hård- och mjukvara som finns i datorn.

Det kommer alltid att finnas säkerhetshål att hitta när provtagare använder egen dator vid provtillfället, och det bästa man kan hoppas på är att göra det svårt att fuska. Vår förhoppning är att de säkerhetsfunktioner som hittats under detta projekt ska göra det avsevärt svårare för en attackerare att kringgå VM-detektionen i företagets SEB-version.

# 7. Slutsats

Delmålen 1–4 hade tydliga krav på vad som skulle åstadkommas och har implementerats framgångsrikt. Dessa delmål är nu integrerade i företagets version av SEB och fungerar tillsammans med företagets servrar.

Delmål 5 hade en mer öppen problemställning och två metoder för att detektera fusk har implementerats. Båda dessa syftar till att detektera virtuella maskiner. Den ena metoden använder datorns TPM-enhet och kontrollerar att den verkar vara legitim. Den andra metoden använder sig av en prestandaräknare som undersöker hur mycket grafikminne som används. Säkerhetshål har identifierats i båda metoderna och dessa beskrivs i rapporten.

Användbarheten av fuskdetektionsmetoderna är inte fastställd då det behövs mer testdata från olika datorer. Dessvärre fanns ej resurser eller tid till att genomföra storskaliga tester. Om de implementerade fuskdetektionsmetoderna fungerar som tänkt kommer de att göra det svårare att fuska för attackerarmodellen och delmålet anses därför vara avklarat. En rekommendation lämnas till företaget att genomföra fler tester och analysera resultatet av dessa innan man börjar använda det i sin slutprodukt.

I delmål 5 undersöktes även Timing Attacks och Modified Client. De publika Timing Attacks som testades ansågs för opålitliga. Att implementera Timing Attacks på egen hand ansågs vara för komplext och tidskrävande för projektet. Modified Client fanns redan i SEB och slutsatsen var att det ger ett tillräckligt bra skydd. När företaget släpper en egen version behöver de ett eget certifikat och applikationen behöver uppdateras till att kontrollera detta certifikat.

Trots att efterforskningar visar att det är möjligt att distribuera en modifierad version av SEB utan att publicera kod som placerats i nya filer lämnas en stark rekommendation till företaget att utreda detta utförligt innan distribution sker.

# Referenser


[1] J. Gjörtler och K. Svensson, ”Ärlighet varar längst? - en studie av elevers och lärares syn på fusk i gymnasieskolan,” 2003. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1498053/FULLTEXT01.pdf. [Använd 17 januari 2024].

[2] ”Mozilla Public License,” [Online]. Available: https://www.mozilla.org/en-US/MPL/. [Använd 4 april 2024].

[3] ”Safe Exam Browser - About,” [Online]. Available: https://www.safeexambrowser.org/about_overview_en.html. [Använd 17 januari 2024].

[4] ”SEB Alliance - Members,” [Online]. Available: https://www.safeexambrowser.org/alliance/members.html. [Använd 17 januari 2024].

[5] ”About GitHub and Git,” [Online]. Available: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git. [Använd 11 juni 2024].

[6] M. De Benedictis, L. Jacquin, I. Pedone, A. Atzeni och A. Lioy, ”A novel architecture to virtualise a hardware-bound trusted platform,” *Future Generation Computer Systems,* vol. 150, p. 23, 2024.

[7] ”Windows 11 - Specifikationer och systemkrav,” [Online]. Available: https://www.microsoft.com/sv-se/windows/windows-11-specifications. [Använd 12 april 2024].

[8] ”MPL 2.0 FAQ,” [Online]. Available: https://www.mozilla.org/en-US/MPL/2.0/FAQ/#:~:text=Other%20Common%20Questions-,Q11,-%3A%20How%20%27viral%27%20is . [Använd 10 april 2024].

[9] ”VBoxHardenedLoader,” [Online]. Available: https://github.com/hfiref0x/VBoxHardenedLoader/tree/630a5c86c2f47d18fa1295cb7c885113bb11ccae. [Använd 27 mars 2024].

[10] ”qemu-patched,” [Online]. Available: https://github.com/kila58/qemu-patched/tree/7adb85977c0d225f641021c2cafbcc9aff3a46c6. [Använd 27 mars 2024].

[11] ”Windows Trusted Platform Module Management Step-by-Step Guide,” 25 juli 2008. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-vista/cc749022(v=ws.10) . [Använd 10 april 2024].

[12] ”TCG TPM Vendor ID Registry Family 1.2 and 2.0,” 11 augusti 2023. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG-TPM-Vendor-ID-Registry-Version-1.06-Revision-0.94_pub.pdf. [Använd 22 februari 2024].

[13] ”Desktop Window Manager,” [Online]. Available: https://en.wikipedia.org/wiki/Desktop_Window_Manager. [Använd 22 mars 2024].

[14] *What happens if you end dwm.exe in different versions of Windows?,* 2021.

[15] Y. Lusky och A. Mendelson, ”Sandbox Detection Using Hardware Side Channels,” i *22nd International Symposium on Quality Electronic Design*, Santa Clara, CA, USA, 2021.

[16] ”Pafish,” [Online]. Available: https://github.com/a0rtega/pafish/tree/b497899ff355ea7b9ecc1f5cd34a9fd1def02aec. [Använd 10 april 2024].

[17] ”Al-Khaser,” [Online]. Available: https://github.com/LordNoteworthy/al-khaser/tree/967afa0d783ff9625caf1b069e3cd1246836b09f . [Använd 10 april 2024].

[18] ”QEMU TPM Device,” [Online]. Available: https://qemu-project.gitlab.io/qemu/specs/tpm.html#the-qemu-tpm-passthrough-device. [Använd 16 februari 2024].

[19] B. Langley, ”GPUs in the Task Manager,” 21 juli 2017. [Online]. Available: https://devblogs.microsoft.com/directx/gpus-in-the-task-manager/. [Använd 1 mars 2024].

**INSTITUTIONEN FÖR DATA- OCH INFORMATIONSTEKNIK**
**CHALMERS TEKNISKA HÖGSKOLA**
Göteborg, Sverige 2024
www.chalmers.se


GÖTEBORGS
UNIVERSITET


CHALMERS