KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Evaluating hardware isolation for secure software development in Highly Regulated Environments

ANDRE BROGÄRD

Stockholm, Sweden, 2023

Evaluating hardware isolation for secure software development in Highly Regulated Environments

ANDRE BROGÄRD

Degree Programme in Computer Science and Engineering
Date: April 17, 2023

Supervisors: Carlos Barreto Suarez, Peter Jonsson
Examiner: Mads Dam
School of Electrical Engineering and Computer Science
Host company: Subset AB
Swedish title: Utvärdering av hårdvaruisolering för säker programvaruutveckling i mycket reglerade miljöer

Abstract

Organizations in highly regulated industries have an increasing need to protect their intellectual assets, because Advanced Persistent Threat (APT) entities are capable of using supply chain attacks to bypass traditional defenses.

This work investigates the feasibility of preventing supply chain attacks by isolating the build environment of the software using hardware isolation. Specifically, this work analyzes the extent to which the Intel SGX can guarantee the integrity and authenticity of software produced in Highly Regulated Environments.

A theoretical evaluation using assurance cases shows that a hardware isolation approach has the potential to guarantee the integrity and authenticity of the produced software. Security weaknesses in Intel SGX significantly limit the confidence in its ability to secure the build environment. Directions for future work to secure a build environment with a hardware isolation approach are suggested. Most importantly, the guarantees from hardware isolation should be improved, suggestively by choosing a more secure hardware isolation solution, and a proof-of-concept of the approach should be implemented.

Keywords

Hardware Isolation, Supply chain attacks, HRE, Intel SGX, CI

Sammanfattning

Organisationer i mycket reglerade industrier har ett ökat behov av att skydda sina intellektuella tillgångar, eftersom avancerade långvariga hot (APT) har förmågan att använda sig av distributionskedjeattacker för att ta sig förbi existerande skydd.

Det här arbetet undersöker möjligheten att skydda sig mot distributionskedjeattacker genom att isolera mjukvarans byggmiljö med hjälp av hårdvaruisolering. Specifikt analyseras till vilken grad Intel SGX kan garantera integriteten och autenticiteten av mjukvara som produceras i mycket reglerade miljöer.

En teoretisk evaluering genom assurans visar att hårdvaruisolering har möjligheten att garantera integriteten och autenticiteten hos den producerade mjukvaran. Säkerhetsbrister i Intel SGX begränsar i hög grad förtroendet för dess förmåga att säkra byggmiljön. För vidare forskning föreslås att garantierna från hårdvaruisolering förbättras, förslagsvis genom att välja säkrare hårdvaruisoleringslösningar, samt att en prototyp av lösningen implementeras.

Nyckelord

Hårdvaruisolering, Distributionskedjeattacker, HRE, Intel SGX, CI

Acknowledgments

I want to thank my supervisor Carlos Barreto Suarez for supporting me through this work. Also, I want to thank the host company, Subset, for being welcoming and inspirational, and especially Peter Jonsson for his support and feedback. I am grateful to my examiner, Mads Dam, for his time and valuable feedback. Finally, I sincerely thank my family and friends who have motivated and encouraged me unconditionally throughout this project.

Stockholm, April 2023
Andre Brogärd

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

APT	Advanced Persistent Threat
CA	certificate authority
CD	Continuous Deployment
CDE	Continuous Delivery
CI	Continuous Integration
CSE	Continuous Software Engineering
GSN	Goal Structuring Notation
HRE	Highly Regulated Environment
Intel SGX	Intel Software Guard Extensions
PCCS	Intel Provisioning Certification Cache Service
PCE	Provisioning Certification Enclave
PCS	Intel Provisioning Certification Service
QE	Quoting Enclave
SDLC	Software Development Life Cycle
TLS	Transport Layer Security

Chapter 1

Introduction

As the technological ecosystem has advanced, modern cyber threats have also become more advanced, and with increased consequences [1]. Moreover, organizations are becoming increasingly aware of the cyber risks and are allocating more resources to cyber security. In a global survey, conducted by various consultancy and insurance firms, cyber risk and data breaches have been identified as the foremost enterprise risk [2]. Interestingly, cyber risk commonly has the property of anonymity because it can remain undetectable until it impacts businesses.

Supply chains, in the form of actors networking together through information technology (IT), are an integral part of the evolving technological ecosystems [2]. Inherently, supply chains are difficult to avoid and it becomes necessary to trust other actors, such as infrastructure providers, monitoring providers, etc., to produce large-scale applications. Every actor and the connections between them pose a potential threat to the supply chain [3]. Efforts in cyber security responses fall behind in relation to the digitalization of supply chains. By collaborating with many partners, supply chains have increased their vulnerability to threats [4].

In 2019, Verizon reported that cyber-attacks through phishing, i.e., gaining access to sensitive information by disguising the threat as a trustworthy entity, are on the rise [2]. The European Union Agency for Cybersecurity (ENISA) identified 24 supply chain attacks between January 2020 and July 2021. As part of their report, they found that in 50% of the attacks, the attackers can be attributed to Advanced Persistent Threats (APTs) [5].

In 2021, Kaseya became aware of a vulnerability in their SaaS platform used by money services providers where attackers could distribute digitally signed malicious software as if it was trusted [6]. In turn, the money services providers' customers, who received the malicious updates, were affected by the attack. The attackers could encrypt their data and launch ransomware claims. It has been estimated that up to 1500 downstream businesses were affected by the attack on Kaseya's platform. Among the affected customers was Coop, a Swedish grocery store, which had to temporarily close around 800 stores.

Similarly, SolarWinds was breached in 2020, where attackers could insert malicious code into the software before it was digitally signed and thus falsely trusted by their customers. More details on the SolarWinds attack are presented in section 2.4.

1.1 Background

Continuous Integration is a common part of the Software Development Life Cycle (SDLC) and is considered software development best practice [7]. When producing software, the software is often tested and built as a part of the CI process. Security is essential in the CI process, as the introduction of malicious code can have large repercussions and enable attacks on consumers of the software.

Highly Regulated Environments (HREs) are software development environments where the software produced is consumed in highly regulated industries, and where there is an increased requirement to protect intellectual assets and prevent cyber threats [8]. Characteristically, there is air-gapped separation between computer systems (offline networks), inability to discuss project details off-premises, and inability to take software artifacts off-premises. HREs, by design, mitigate many direct cyber threats, but indirect attacks are still possible. Albeit, it requires more resources from an attacker.

In recent years, efforts have been made to secure the CI process further. Muñoz et al. [9] used a form of hardware isolation to ensure project integrity throughout the process of CI; however, no consideration was taken to modifications in the build process itself, for example, if the result of building the software contains malicious files. Reproducible builds are desirable, as they enable external or internal parties to verify the software's integrity [10]. Hardware isolation is a commonly used approach to protect the execution of software by leveraging CPU hardware. This approach guarantees both the confidentiality and integrity of the executing software [11]. Hardware isolation solutions, such as Intel SGX, AMD SEV, and ARM TrustZone, could make it possible to secure the CI process, especially if the software builds are non-reproducible.

Compared to reproducible builds, trust is placed in a single system provided by the hardware manufacturer [12]. While hardware isolation offers benefits, it is essential to consider the potential limitations of this approach. For instance, hardware isolation may not be suitable for all types of software development environments and applications. Therefore, it is necessary to evaluate the limitations of hardware isolation as applied to the CI process.

1.2 Problem

A gap exists between existing efforts to secure CI processes, as well as their applicability in HREs. Furthermore, the CI process requires more consideration and additional guarantees to mitigate supply chain threats. In the case of the Kaseya and SolarWinds attacks, digital signatures are used to ensure that the origin of the software is valid. However, the trust in the digital signature must be strengthened. Specifically, the integrity of the produced software must be ensured in order to prevent threats from propagating in the supply chain. However, when reproducible builds are not possible, there is a greater need to place trust in the system that must build the software and ensure that no build artifacts are modified.

To the author's best knowledge, this work takes a different approach to securing the CI process than what has been found in previous works. Unlike the case of reproducible builds, in this work, trust is placed in a single system. Additionally, this work takes the approach of isolating the entire CI process, including the build process. The main research question in this work is:

- To what extent can hardware isolation be used to secure a Continuous Integration process in the context of a Highly Regulated Environment?

To answer the research question, this work proposes and evaluates a hardware isolation approach to the CI process that applies to HREs, where the software artifacts produced are verifiable and trusted to a similar level as the hardware isolation guarantees can be trusted. Furthermore, an analysis of the proposed approach and directions for future work, to better answer the research question and strengthen the verifiability of produced software artifacts, is presented.

1.3 Purpose

Threat actors are becoming increasingly sophisticated, and the CI process poses a large cyber risk because it can cause significant harm to an organization and its customers if it is compromised. Therefore, organizations will benefit from reducing the cyber risk in the CI process. For organizations that are involved in highly regulated industries, reduced cyber risk is important and, by requirement, they may need to verify and present a convincing argument that the software produced has not been modified.

Security researchers constantly find vulnerabilities and exploits in operating systems and applications [12], which makes it difficult to rely on only software as protection for the CI process. Hardware isolation, specifically Trusted Execution Environments, has been applied to safety-critical applications before [11].

1.4 Goals

The goal of this project is to explore and analyze a hardware isolation approach to a secure and verifiable CI process that can be used to strengthen the trust that can be placed on digital signatures of software in the supply chain. This has been divided into the following two sub-goals:

1. Present an approach that would allow organizations to reduce cyber risk in their CI process, especially applicable to organizations in highly regulated industries. Furthermore, the approach should contribute to mitigating supply chain attacks.

2. Guide future research on the application of hardware isolation on CI processes.

1.5 Research methodology

This work aims to evaluate the feasibility of hardware isolation in Highly Regulated Environments and provide a theoretical assessment of the proposed approach's weaknesses and potential strengths.

To evaluate the feasibility of a hardware isolation approach to the CI process, several key elements were defined. First, a threat model on the CI process was established using the STRIDE framework to assess potential attacker goals and capabilities. Next, the security requirements for the proposed approach were defined to ensure a secure CI process. Additionally, the properties of the hardware isolation technology, in this work Intel SGX, were presented, and assumptions related to the proposed approach were established.

To evaluate the hardware isolation approach, inductive argumentation was used with a qualitative measurement of sufficiency. Safety cases, expressed through Goal Structuring Notation (GSN), were used to structure the argumentation. The goal is to establish assurance that the security requirements of the proposed approach hold.

1.6 Delimitations

This work focuses on HREs, which have particular requirements. For example, security solutions must fit air-gapped environments. Likewise, the threat model is restricted to APTs, which are the type of threat actors that may target HRE (more details about the security context are presented in chapter 4). Furthermore, security is prioritized over usability in the proposed CI approach. Because of time restraints, no prototype is produced as a proof-of-concept. Instead, this work presents a theoretical CI approach.

1.7 Structure of the thesis

Background and related work to support this work are introduced in chapter 2. The research method, including the evaluation framework, is described in chapter 3. The framework design of the CI process using hardware isolation is presented in chapter 4. The framework is evaluated in chapter 5 and the result of the evaluation is discussed in chapter 6. Finally, the conclusions and directions for future work are presented in chapter 7.

Chapter 2

Background

This chapter presents areas relevant to the context and methodology of this work, such as the Software Development Life Cycle (SDLC) in section 2.1, qualities and methods for ensuring secure applications in section 2.2, Highly Regulated Environments (HREs) in section 2.3, and supply chain attacks and their characteristics in section 2.4. Additionally, the main properties of Intel Software Guard Extension (Intel SGX) are presented in section 2.5, which is the underlying technology this work is based on. Moreover, section 2.5.4 presents a security analysis of Intel SGX. Finally, related work is presented in section 2.6, and a summary of the background is available in section 2.7.

2.1 Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a framework that defines the processes of software development from the conceptualization of the product to deployment and maintenance [13]. An SDLC is either defined as linear, iterative, or a combination thereof. Development requirements are expected to change during development, and the different SDLC models handle these changes in different ways. Also, SDLC models vary depending on the environment in which they are used and the context for their application. Commonly, an iterative approach is used. As an example, and to highlight the different stages of a common SDLC, the following are phases of the Waterfall model, which was defined in 1970: evaluation, requirements, analysis, design, development, validation, and deployment [13].

The waterfall model includes feedback loops to ensure that phases can be revisited. Although there are more modern models than the waterfall model, it can be regarded as a baseline model containing characteristic phases of a general SDLC model.

The scope of the SDLC used in this work is limited by the area of Continuous Software Engineering, which is further detailed in the following section.

Continuous Software Engineering

To the SDLC, Continuous Software Engineering (CSE) improves mainly on the steps characterized as development, validation, and deployment, as well as enabling the need for rapid changes to requirements in these steps efficiently. CSE is an area of both research and practice [7]. It includes three phases: Business Strategy and Planning, Development, and Operations.

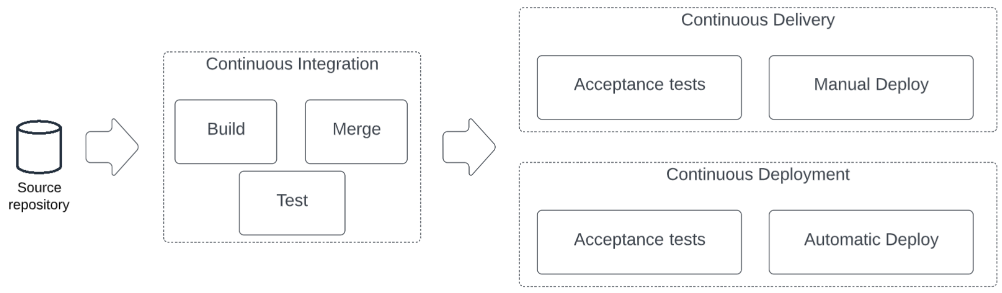

Figure 2.1: The CI, CDE, and CD practices of Continuous Software Engineering. Adapted from [7].

Continuous Integration (CI), Continuous Delivery (CDE), and Continuous Deployment (CD) are practices of CSE that are used to make the Development and Operations process as efficient as possible [7]. In this study, special consideration will be given to CI because that is often where the software is built. See figure 2.1 for an illustration of the practices.

Continuous Integration (CI).
CI involves practices for automated processes for building, merging, and testing development work (such as source code) that originates from a source repository, such as Git. It enables organizations to have more frequent code releases, improves the software quality, and productivity [7].

Continuous Delivery (CDE).
CDE involves practices for ensuring an application is production-ready. CI is an important component in CDE and precedes further acceptance tests and deployment automation.

Continuous Deployment (CD).

CD involves practices for Continuous Deployment and delivery of applications. CD implies CDE, but the reverse is not true [7]. The main difference is that in CDE, it is typical that a manual step, or a separate process, is necessary to decide when the application is deployed to a production environment (with active customers). In CD, applications are commonly deployed continuously to the production environment as new changes are merged and tested in the preceding CI process.

2.2 Secure applications

Producing secure applications requires a secure design and implementation. Additionally, applications need to be deployed in a secure platform, and their security needs to be motivated [14]. Below, these items are described further.

Secure platforms are essential to provide both the application and users with trusted security mechanisms to fulfill their security requirements.

Secure design utilizes security patterns and models, commonly established by experts, that fulfill particular requirements on security. Usually, many security patterns and models need to be applied to fulfill all security requirements.

Secure implementation is difficult to formally verify for complex applications. However, a secure design can compensate for the lack of formal verification methods. Additionally, several development practices of the SDLC can be used to reduce the risk of poor implementations that compromise security.

Security assurance is used to motivate the security of an application through structured informal arguments, also known as safety or assurance cases. It is extensively used in the safety-critical system community [15].

In sections 2.2.1 and 2.2.2, desirable properties and the STRIDE threat model are presented, which are fundamental to creating a secure design. Next, in section 2.2.3, security assurance is described further. Safety cases and notations for presenting them are also introduced.

2.2.1 Desirable properties

The CIA-triad refers to fundamental security properties in information systems and has been around since the 1970s [16]. In this work, the CIA-triad will be used extensively to explain the guarantees of relevant technologies. The CIA-triad is often used as a reference point when discussing secure design. It is an acronym for three security properties [17].

- Confidentiality (C) - Prevention of unauthorized disclosure of data.
- Integrity (I) - Prevention of unauthorized modification of data.
- Availability (A) - Prevention of unauthorized withholding of data.

Academically, the CIA-triad has received criticism for being narrow in its scope and not including social and non-technical challenges in security. The CIA-triad has recognizably been extended with more properties during the 1980s and 1990s, such as non-repudiation (nR) and authenticity (Au) of information [16].

- Non-repudiation (nR) - The inability to defy a certain transaction or communication between two parties. For example, if Eve sends a malicious packet to Bob, then Eve is unable to later deny the action and claim that she is not responsible for the malicious packet sent to Bob.
- Authenticity (Au) - The quality of being original and genuine. Confidence that an identifier is associated with an item or person.

The definition of the CIA-triad and its extensions are continuously evaluated and redefined in academia. This work will use the definition and extended properties as presented above.

2.2.2 STRIDE threat model

Threat modeling can be used to detect flaws in software designs already in the design phase, which allows iterations of necessary security requirements [18]. It is also recognized as one of the most important activities in software security.

STRIDE is an acronym for six categories of threats [18]:

- Spoofing (S) - An entity (person or program) can impersonate a legitimate entity.

- Tampering (T) - An attacker can illegitimately modify application resources, such as data.
- Repudiation (R) - A user (legitimate or malicious) can deny actions within the system.
- Information Disclosure (I) - An attacker can obtain private data.
- Denial of service (D) - An attacker can make the system unavailable to other users.
- Elevation of privilege (E) - An attacker can obtain privileged access to protected resources.

The STRIDE threat model is used in this work to analyze the approach to CI using hardware isolation presented in chapter 4.

2.2.3 Safety cases

The defence standard 00-56 [19] (2007) defines safety cases as "a structured *argument*, supported by a body of *evidence*, that provides a compelling, comprehensible and valid case that a *system is safe* for a given application in a given environment". Safety cases are particularly used to provide an assurance viewpoint to demonstrate that security properties have been satisfied and that risks have been mitigated [15]. The purpose of a safety case is further summarized in [20] as "a safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context".

In [15], Bloomfield et al. offer their perspective on safety and assurance cases and present the following three approaches:

- Use of accepted safety standards and guidelines to demonstrate compliance.
- Goal-based. Justification via a set of claims/goals about the system's safety behavior supported by evidence.
- Vulnerability-based. Investigation of known potential issues of the system, which is more of a bottom-up instead of a top-down approach.

Strong arguments should be both valid and sound [20]. Valid arguments imply that if the premises are true, the conclusion is true. Sound arguments have true premises.

Goal Structuring Notation

GSN was introduced in [20] and is a graphical argumentation notation representing elements of any safety argument (such as goals, evidence, and arguments). Although there is a desire to have both provably sound and valid arguments, it is unobtainable for a safety-critical system because of its complex nature. Therefore, arguments in GSN allow uncertainty - if premises are true, the conclusion may be true - which is a weaker form of argument.

Uncertainty in arguments for assurance in safety cases leads to a form of inductive argumentation: if premises are true, the conclusion is true with some probability associated [20]. The purpose of the argument becomes to sufficiently demonstrate that certain security goals are met.

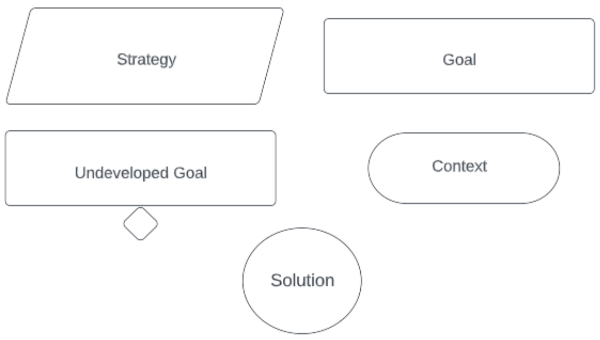

Figure 2.2: Principal elements of GSN

Figure 2.2 presents the different principal elements in GSN [20] as they visually appear in this work. Goals represent what the assurance case wishes to accomplish, and can be divided into interlinked sub-goals. Strategies provide details about how goals or sub-goals should be fulfilled from their respective sub-goals. Contexts limit the scope of a strategy or goal. Solutions present a reason as to why certain goals are fulfilled. Undeveloped goals have no solution or are otherwise excluded from the assurance case. Each principal element type is indexed individually.

A high-level example of GSN can be seen in figure 2.3, which was inspired by more detailed examples used in the original report. The GSN aims to fulfill goal G1, that the application is not vulnerable to any threats. The strategy used, St1, is to omit all threats, and for each threat, a goal is defined indicating that the application is not vulnerable to the threat. The context, in this case, is C1, which defines all known threats. Goals G2 and G3 can be fulfilled individually.

The solution in this example, S1, is black box testing, which should fulfill both G2 and G3. By fulfilling G2 and G3, G1 is also fulfilled in context C1.

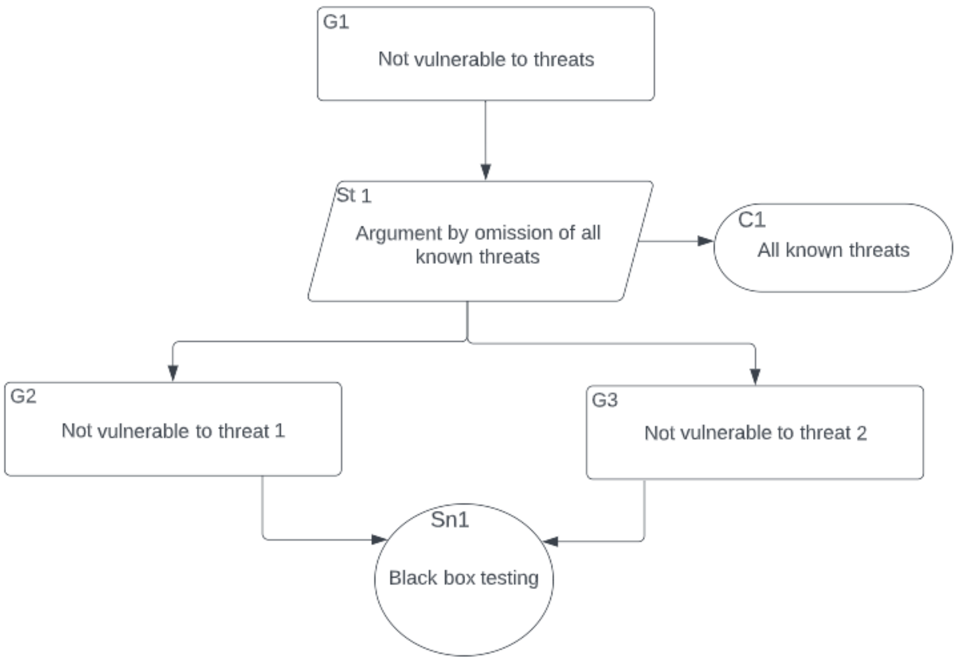
G1
Not vulnerable to threats
St 1
Argument by omission of all known threats
C1
All known threats
G2
Not vulnerable to threat 1
G3
Not vulnerable to threat 2
Sn1
Black box testing

Figure 2.3: A GSN safety case example

In this work, a goal-based approach to safety cases and assurance is used. Safety cases are structured by their claims, evidence, and arguments. The arguments presented are inductive with a qualitative measurement of sufficiency.

2.3 Highly Regulated Environments

In [21, 8], Morales et al. presented characteristics of Highly Regulated Environments (HRE) as air-gapped physical spaces and computer systems with heightened security and access control, segregation of duties, the inability of personnel to discuss certain topics outside specific areas, and the inability to take certain artifacts, such as code, software dependencies, etc., off-premises. HREs are also different on a case-to-case basis [8]. Usually, the policies surrounding the project determine the extent of the HRE and exactly what is enforced.

If not using air-gapped environments, the approach used to protect sensitive information is behind several layers of routers, firewalls, access control, and more. Despite these efforts, it is difficult to assure that no breach can occur and that information cannot be transmitted to the outside world if the network is connected to the internet. Here, organizations can resort to air-gapped environments, where the network on which the sensitive assets are placed has no connection to the internet [22]. This eliminates the trust organizations must put on configurations of routers and firewalls and makes attacks more difficult. It is a common characteristic of an HRE.

HREs are primarily used in organizations that operate in or are involved in highly regulated industries. Such industries include the military or defense and could also involve chemical and banking industries and more. For instance, the Swedish Försvarsmakten (*eng:* The Swedish Armed Forces) has issued documents regarding requirements and security functions (KSF) [23] for IT systems that are developed for Försvarsmakten. A significant portion of the requirements and security functions specified in KSF can be fulfilled, partly or in full, by operating in an HRE.

2.3.1 SDLC obstacles in HREs

The characteristics of HREs imply several limitations on the SDLC and how it can be implemented in an organization. Moreles et al. [21] identified some properties of HREs that make it difficult to guarantee the security requirements of the SDLC. Two of these obstacles are presented here:

1. Air-gapped environment
2. Partial or no production environment access

Air-gapped environments typically isolate the development personnel from the technology, which limits aspects of the SDLC, such as cooperation and feedback from stakeholders. The development team may lack access to the production environment partly or in full. Parity between environments is important as it could introduce further security vulnerabilities if applications do not function as intended in their environment.

In this work, development in an HRE is defined to include the following constraints: air-gapped environments per project, the inability of personnel to discuss projects off-premises, the inability to take development artifacts outside the environment, and strict policies to bring artifacts into the environment. Additionally, the SDLC in the HRE excludes automated deployment of software; instead, it produces a deliverable in the form of software and there is no production environment access for the development personnel.

2.4 Supply chain attacks

A supply chain refers to the network of *suppliers* involved in the creation of a product. In this context, products are in the form of software or applications that are delivered, or exposed, to *customers*. There can naturally be a long chain of multiple suppliers of software, involved in a specific application that reaches the customer. A supply chain attack occurs when an attacker targets single or multiple customers by attacking a single supplier [5]. This type of attack is commonly exploited by Advanced Persistent Threat (APT) entities.

Ahmad et al. [24] provide a formal definition of APTs: "An entity that engages in a malicious, organized, and highly sophisticated long-term or reiterated network intrusion and exploitation operation to obtain information from a target organization, sabotage its operations, or both." Furthermore, the authors present two forms of APTs: those who are strategically motivated, and those who are operationally motivated. Strategically motivated APTs tend to be part of a larger organized crime network and consist of professional hackers for the benefit of a third-party. Operationally motivated APTs, however, tend to be hackers who engage in criminal activities for personal financial gain.

SolarWinds attack

SolarWinds was breached in early 2020 [25]. The attack, attributed to the Russian state-actor group with nicknames *Cozy Bear* or *APT29*, inserted malicious code into Orion, SolarWinds' monitoring and management platform, which in turn was installed by multiple customers [26]. The customers received the malicious update, which appeared valid and signed by SolarWinds. Consequentially, the attackers could perform reconnaissance and monitor their surroundings to perform a more careful attack directly on multiple customers. The attack itself has been deemed highly sophisticated, partly because it was designed to avoid detection throughout its lifetime. If it determined that the risk of detection was too high it would delete itself; otherwise, it would continue to install more malicious code. Another aspect of its sophistication was the customers it targeted and whom it chose to compromise.

Among the affected customers are Microsoft, the U.S. government, VMWare, DUO, and FireEye [27]. The attackers have been reported to have obtained source code from Microsoft (but Microsoft denies breaches due to the attack). The U.S. government reported having had certain emails stolen. FireEye, a cybersecurity company who also were first to discover the attack, had its cybersecurity tool suite stolen. Several targets from the SolarWinds attack seem to have been organizations such as Microsoft and DUO that deliver services to many businesses to best increase the effectiveness of the attack.

The attack originating from SolarWinds exploited the trust that firms such as Microsoft and the U.S. government put in their suppliers. Tools already in place to detect sophisticated attacks were insufficient to detect it. The same vulnerability characterizes supply chain attacks in general [5].

2.5 Trusted Execution Environments and the Intel Software Guard Extension

Most applications today are executed on a large untrusted computing base, for instance, the operating system, hypervisor, and firmware. The untrusted computing base is often unverifiable [11]. Trusted Execution Environments (TEEs) provide a trusted computing base, which is a secure area that can be used to run security-critical applications separately from untrusted applications. TEEs rely on hardware and software features to ensure the security of the applications [11].

Intel Software Guard Extension (Intel SGX) is a trusted computing design aimed at solving the problem of secure remote computation [12]. Secure remote computation is generally unsolved. Existing solutions, such as fully homomorphic encryption, have significant impractical aspects.

Intel Software Guard Extensions (Intel SGX) was chosen over other Trusted Execution Environments for this work because of its third-party attestation scheme, which is detailed later in this section and allows it to be used offline. Alternatives include, but are not limited to, ARM TrustZone, AMD SEV, and Sanctum [28, 12, 29]. How Intel SGX compares to alternatives is detailed more in section 2.5.5.

Only the main properties of Intel SGX relevant to the work are included in this section. In the following subsections, the architectural overview of Intel SGX along with the considered attestation scheme, the main concepts of how a secure container can verify itself to other users, and a security analysis and threat model of Intel SGX.

2.5.1 Intel SGX attestation

More precisely, the attestation scheme used in this work is based on the Elliptic Curve Digital Signature Algorithm (ECDSA). The main goal of attestation is to prove to a user that they are communicating with a pre-determined application, running in a secure SGX container [12].

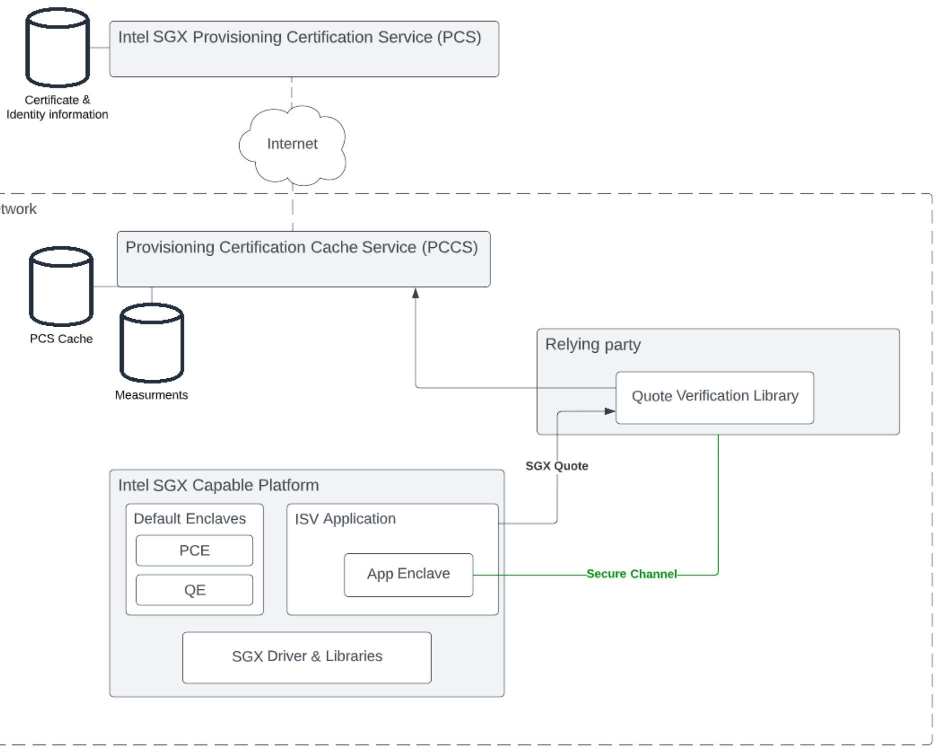
Intel SGX Provisioning Certification Service (PCS)
Certificate & Identity information
Internet
Network
Provisioning Certification Cache Service (PCCS)
PCS Cache
Measurments
Relying party
Quote Verification Library
SGX Quote
Intel SGX Capable Platform
Default Enclaves
PCE
QE
ISV Application
App Enclave
Secure Channel
SGX Driver & Libraries

Figure 2.4: Simplified Intel SGX setup. Adapted from [30]

In figure 2.4, the overall architecture and the relationship between components are visualized. This section will clarify each component's purpose for a trusted computing design, as well as how they are related to the attestation process.

The attestation scheme was introduced by Intel to allow third parties to build their own attestation services [31]. Additionally, it supports attestation to be carried out in an offline network if configured accordingly.

Components

Secure containers in Intel SGX are called *enclaves*. Enclaves run inside an *SGX Platform* [30]. The developers of the enclaves are called Independent

Software Vendors (ISVs).

An SGX Platform, a machine with an Intel SGX-enabled CPU configured to act as an SGX Platform, is at manufacture time provisioned with keys only known to Intel [12]. In this work, the keys are referred to as *provisioned keys*. The *provisioned keys* are used to identify the SGX Platform.

In each SGX Platform, two Intel-issued enclaves are necessary for attestation, namely the Provisioning Certification Enclave (PCE) and Quoting Enclave (QE). The PCE uses the provisioned keys, known to Intel, and the QE instead uses a generated *attestation key*, which is unknown to Intel and other participants [30].

When an enclave is loaded into the SGX Platform, the enclave initiates the creation of an SGX Report. The SGX Report contains information about the application itself and its environment, also called a *measurement* or *measurement hash*. The QE is responsible for signing the SGX Report, which produces an SGX Quote. The SGX Quote is shared with other parties and is used to verify the application [12].

The Intel Provisioning Certification Service (PCS) is an Intel-provided service that can identify genuine SGX Platforms. Additionally, it holds necessary information to verify SGX Quotes, such as allowed measurements and the identity of genuine SGX Platforms and Intel-issued enclaves [12].

Intel Provisioning Certification Cache Service (PCCS) can cache results from the PCS. The PCCS is what allows the attestation to be performed in a closed network as verification libraries used by users in the system can query the PCCS for the necessary information, rather than using the PCS [30]. It is up to an administrator to keep the PCCS updated if it is configured to be offline.

By using the components above, a user in the system (see the “relaying party” in figure 2.4) can establish a secure channel to communicate with the application in the SGX Platform.

2.5.2 Chain of trust

Trust in the identity of SGX Platforms and the genuineness of SGX Quotes is rooted in the general certification infrastructure set up by Intel. This section will briefly present the fundamentals of Public Key Infrastructure (PKI), which is the basis for the general certification infrastructure, and how the trust is propagated through components in the overall architecture of SGX.

PKIs allow entities to prove their identity to each other [32]. X.509 is a standard for PKI. certificate authorities (CAs) issue certificates that bind an identity to a public key, achieved using digital signatures. The certificates themselves contain information about the identity and the associated public key. Additionally, CAs can issue intermediary certificates, that allow the recipients of such certificates to issue certificates to other entities. CAs are responsible for ensuring the identity of the entity that is issued a certificate. The tree of certificates, rooted in the certificate held by the CA, is commonly referred to as a trust chain. Furthermore, certificates can be self-signed, which is the case for root certificates. Identities of self-signed certificates do not have to be verified by a CA.

The PCCS stores certificates for the provisioned keys used in the PCE, which are signed by Intel's root CA. The PCE will sign the QE's attestation key, which in turn will sign the SGX Report and produce an SGX Quote. This certificate structure is verified during attestation. The trust in the SGX Platform is rooted in Intel's root CA [30, 12].

Importantly, trust that the application is the pre-determined application is also dependent on Intel SGX's guarantees of enclave security. These guarantees are a part of the Intel SGX threat model, which is presented in section 2.5.4.

2.5.3 Application of Intel SGX

The implementation of hardware isolation through Intel SGX is mainly to provide a semi-sandbox mechanism, in which the sensitive and private parts of the application are placed within the enclave, such as nontransparent code. In this manner, SGX provides a secure environment for executing sensitive applications [11].

Several practical applications of Intel SGX have been explored, including network defense mechanisms for web browsers, data security in databases, improvement of data encryption technology, protection of sensitive data in computer games, and data security in Bluetooth I/O processes. Intel SGX has also been applied to cloud computing, for instance, where it has been used to provide more secure memory for big data applications [11].

2.5.4 Threat model

In Intel SGX's threat model, all non-enclave resources; such as the operating system, the hypervisor, applications, and peripherals are considered untrusted [12]. Within enclaves, the following guarantees are part of the threat model:

- Confidentiality - To prevent reading code and data within enclaves.

- Integrity - To prevent modification of code and data within enclaves.

Furthermore, to prevent replay attacks (i.e. unauthorized re-use of data), the computation within enclaves also has freshness guarantees [12].

In [12], Costan et al. analyze and evaluate Intel SGX with respect to different types of attacks, it is also compared to other hardware isolation technologies. The comparison to other hardware isolation technologies is summarized in section 2.5.5. Next follows a categorization of attacks and Intel SGX's resilience to these types of attacks.

2.5.4.1 Attack categorization

The attack types considered for Intel SGX are shown and described in table 2.1. The attack types are also divided into three categories as in [28]: attacks from privileged software, attacks from hardware events, and attacks from hardware probing (physical access to the hardware). Intel SGX's resilience to each attack type is presented in section 2.5.4.2.

Category	**Attack type**	**Description**
Privileged Software	Direct memory access (DMA)	Transferring unauthorized data from memory to peripherals without involving the CPU [12]
	Address translation	Passive: A malicious system can infer information from memory access patterns. Active: A malicious system can modify page tables such that it breaks the virtual memory abstraction [12].
	Firmware	Firmware can be overwritten by system hardware. Untrusted system hardware can inject malicious code into the firmware.
	Denial of service	Preventing an enclave from executing.
Hardware events	Cache-timing	Exploits the dependency between the location of memory access and the time it takes to perform the operation. Can be mounted by unprivileged software.
	Row-hammer attack	A class of attacks where some DRAM (memory) hardware implementations are vulnerable to bit-flipping attacks from unprivileged software.
Hardware probing	Port attacks	Physically accessing the ports on the processor chip, for instance, the debug port.
	Bus tapping attacks	Physically accessing the memory bus, reading and analyzing data.
	Chip attacks	Physically accessing the processor chip and interacting with its electrical circuits.
	Power-analysis attacks	Measuring the power consumption of a computer system or its components.

Table 2.1: Description of different attack types related to Intel SGX

2.5.4.2 Resilience

Table 2.2 describes the resilience of Intel SGX to the attack types in table 2.1 [28, 12].

Attack type	**Resilience**
Direct Memory Access (DMA)	Secure, because DMA access is rejected by reserved enclave memory.
Address translation	Not secure against the passive type. Secure against active types of address translation.
Firmware	Secure, because firmware can not circumvent access checks.
Denial of service	Not secure, because enclaves execute on top of untrusted system software.
Cache-timing	Not secure, it is possible to evict the desired cache line.
Row-hammer attack	Secure, integrity checks can detect unexpected modifications of data.
Port attacks	Secure, unless debug ports are enabled.
Bus tapping attacks	Secure, because the CPU encrypts data in transit.
Chip attacks	Not secure.
Power-analysis attacks	Not secure

Table 2.2: Intel SGX resilience to different types of attacks

The authors of [12] conclude that Intel SGX is resistant to straightforward attacks, but the resistance is almost non-existent for sophisticated attacks such as cache-based side-channel attacks, and most hardware probing attacks. Furthermore, the authors found that several security-critical aspects of Intel SGX lack rigorous documentation, making it guesswork to argue for some of its security claims.

2.5.5 Alternatives to Intel SGX

This section presents the main differences in ARM TrustZone and Sanctum to Intel SGX, both concerning their approach and scope the of threat model.

ARM TrustZone. ARM TrustZone conceptually partitions a system into a secure world and a normal world. Intel SGX instead encapsulates software in an enclave [11]. ARM uses semiconductor intellectual property cores (IP blocks) to separate the system's resources (CPU, memory, peripherals). Each world has its own operating system and software resources. ARM TrustZone can protect against passive address translation attacks, unlike Intel SGX. However, similar to Intel SGX, it fails to protect against cache-based side-channel attacks, such as the cache-timing attack [28].

Sanctum. Sanctum's security argument relies on two pillars: enclave isolation enforced by a security monitor, and guarantees of the software attestation signatures [29]. Sanctum increases the scope of Intel SGX's threat model by including attacks that analyze an enclave's memory access patterns, especially cache-timing attacks, because they can be launched from unprivileged software. The prototype targets a Rocket RISC-V core and is open-source.

2.6 Related work

In this section, relevant attacks and contributions to HREs and CI are presented. Additionally, this section presents open-source contributions to Intel SGX.

2.6.1 HRE attacks

Air-gapped environments significantly limit the possible attacks on HRE environments. Research into attacks on HREs has assumed that malicious code already exists on the compromised computers [22, 33].

In [33], the authors demonstrated how information can be leaked from an air-gapped computer by transmitting information through flashing LEDs on the keyboard. The same vulnerability can be used for any visible and controllable light source on the computer. An attack could for instance use drones to record and receive the transmission.

In [22], data was exfiltrated by controlling the CPU utilization. The attacker is assumed to have access to the electrical socket that powers the computer and can listen to fluctuations.

2.6.2 Secure CI

In [14], Rimba. et al. propose using verifiable design fragments as a security pattern for capability-based platforms, such as those provided by AWS IAM where permissions are explicitly stated by relations of actions on resources for a particular user, and a pattern-based composition approach to build and verify an application design on a capability-based platform. Their verifiable design fragments and pattern-based composition approach allow to create security assurance for the application in the design phase. In their case study, they apply the approach to build a secure CI/CDE/CD pipeline on AWS and divide the pipeline into trusted and untrusted resources that make up the design fragments.

In [9], Muñoz et al. proposed and demonstrated a framework for ensuring project integrity throughout the phases of fetching source code and dependencies and through the build and testing process using Trusted execution environments (TEE) and Secure elements (a chip designed to prevent unauthorized access and store confidential and cryptographic data). However, integrity checks throughout phases cannot assure that the build output is valid in those cases where the build output contains new files or binaries.

Gruhn et al. [34] targeted an environment where a Jenkins server executes builds in a multi-tenant environment. They only target attacks where one tenant may compromise the CI pipeline of another tenant. Through virtualization, and restoration of each tenant's config through snapshots, this attack was considered infeasible.

2.6.3 Intel SGX open-source contributions

The Gramine project [35] is an open-source lightweight library operating system that runs on Intel SGX. It supports both forms of Intel SGX attestation methods. It allows a developer to simulate a virtualized environment on Intel SGX and run arbitrary applications.

Intel does not define a standard protocol to use for communication between the ISV and the enclave after attestation. Knauth et al. [36] introduced RA-TLS, which extends the Transport Layer Security (TLS) protocol with Intel

SGX attestation information. This allows the remote party to be sure they are communicating with an enclave. RA-TLS supports both forms of attestation methods. Furthermore, Gramine has incorporated this TLS protocol for use with Intel SGX [35]. For the attester to trust the TLS certificate, RA-TLS binds the credentials used for TLS to the SGX Quote produced in the SGX Platform by including a hash corresponding to the credentials in the SGX Quote.

2.7 Summary

CI and CDE are commonly used processes for building and testing applications. Supply chain attacks cause severe damage to organizations and their customers. Characteristically of supply chain attacks, attackers can exploit the trust that customers put in their suppliers. As can be seen from previous attacks, attackers can often tamper with the building and testing of applications undetectable to the supplier.

Highly Regulated Environments and Intel SGX have been introduced, which both are state-of-the-art practices and methods to help mitigate attacks. HREs enforce producing applications with a smaller attack surface and introduce limitations in how applications can be produced. Intel SGX provides an execution environment that claims to guarantee confidentiality and integrity. The trust that the guarantees hold is rooted in Intel. The guarantees are criticized in the literature as they do not hold against sophisticated attacks.

Chapter 3

Method

3.1 Research process

The main purpose of this thesis is to harden the security of the CI process in an HRE environment. The research method is based on the goals of this thesis, which include guiding future research on the application of hardware isolation on CI processes and providing a more comprehensive understanding of the theoretical foundations and the risks of using hardware isolation for the CI process. The goal of this thesis is not to provide a security analysis of any hardware isolation technology.

The technological background was obtained from a literature review on related areas and especially Intel SGX. Intel SGX was chosen over other Trusted Execution Environments because of its third-party attestation scheme, which can be used offline and fulfills this essential requirement for Highly Regulated Environments. Furthermore, open-source contributions such as Gramine and RA-TLS simplify any attempts to implement the proposed framework in this work for practical use.

A baseline CI process and threat model using STRIDE is described for a system without hardware isolation in section 3.3. The framework design described in chapter 4 applies Intel SGX to a similar CI process, and the threat model is adjusted for the new system.

The framework is primarily designed to ensure the integrity of execution and data in the CI process to mitigate supply chain attacks. As a second priority is the confidentiality of the CI process, which may process secrets and is important in an HRE. The framework design's requirements were selected based on the integrity and confidentiality requirements of the CI process.

The framework design was evaluated in chapter 5 based on the security requirements and the adjusted threats. The evaluation framework is described in the following section.

3.2 Evaluation framework

Goal Structuring Notation is used to demonstrate goals and sub-goals and how they relate to strategies, contexts, and solutions as part of a safety case. The evidence, which here is used as the solutions, will be in the form of arguments based on the assumptions mentioned in section 4.8 and on choices made in the framework design.

Subgoals are identified from the choice of strategies and contexts that are included in the Goal Structuring Notation. How strategies and contexts are chosen is further motivated in section 5.1.1.

The security requirements in section 4.9 are transformed into goals (G1, G4, G9, G12) and their respective sub-goals. The evidence is omitted from the graphical part of the Goal Structuring Notation. The evidence is in the form of arguments based on the assumptions and choices made in the framework design.

The Goal Structuring Notation was chosen over a simpler approach, such as simply describing the sufficiency of fulfilling the security requirements. With this method, the arguments and goals are visibly structured and partly explained by the principal elements in GSN. Furthermore, goal-structured safety cases facilitate breaking down the complexity of the approach into smaller, manageable components.

The validity of claims - if premises are true, the conclusion is true - and their soundness - if a valid argument has true premises - are both evaluated. To evaluate the soundness, a qualitative confidence level is associated with each assumption based on findings in previous work. Furthermore, the sufficiency of argumentation in the safety case is discussed. The goal is that this evaluation method can demonstrate the theoretical feasibility, weaknesses, and potential strengths of the proposed approach.

3.3 Baseline CI model

For reference, a general CI model is included here. It will be adjusted further in section 4.7 to the introduced framework design. The general CI model used in this work excludes any form of process for deployment, neither manual nor automatic, for the given application. This is because of two reasons:

HREs rarely contain automatic procedures for deployment, and the manual approach could be complicated. The primary scope of this system is building and validating the software. Therefore, this system is constrained to the CI process.

CI setup

The attack vector is defined on a trivial CI setup in an air-gapped environment (i.e. offline network). The following entities are considered part of the CI setup:

- A worker. Responsible for building, testing, and packaging the software.
- A user. Responsible for instructing the worker on what to execute and providing necessary artifacts such as code or secrets.

The worker and the user can co-exist on the same machine but for the purpose of resembling the framework proposed in chapter 4, and with no difference to the attack threats, the worker and user exist on separate machines within the same closed network.

As a part of the CI pipeline, the user issues jobs describing what to build to the worker, possibly through some third-party and open-source software. The jobs executed inside the worker fulfill the function of different stages of the CI pipeline. The worker may run the job in a virtualized environment, or directly on the host.

The user provides all necessary data, such as code, secrets, and other dependencies to the worker as part of the job description. The worker executes the job and returns the result.

Attacker assumptions and capabilities

The attackers are highly sophisticated and can dedicate a large amount of resources to conduct an attack. Sophisticated APT groups or state-sponsored attacker groups may find that certain HREs are a prioritized target.

By the characteristics of an HRE, attacks are more difficult and often rely on human participation, accidental or deliberate, to inject the exploit into the closed system. It is assumed that every device/machine in the network can be compromised with elevated privileges, but every machine is not necessarily compromised.

Attack threats

The attack scenarios are centered around STRIDE threats. In this scenario, an attacker is assumed to have achieved elevation of privilege on either machine (user or worker machine). The attackers are assumed to have root access on machines they have compromised.

The following STRIDE threats are considered for the compromised worker machine:

- Tampering. An attacker with full privileges to the worker machine can read and modify the memory of applications that are running. They can also tamper with the file system. This may allow the attacker to inject or remove data as part of the job result that is unknown to any other party.
- Information disclosure. The attacker can inspect and extract data passed to and produced from the worker machine.
- Repudiation. The user has no guarantees that the worker will execute the job as detailed and while preserving integrity. Non-repudiation cannot be guaranteed.
- Spoofing. The attacker could imitate the worker and hijack communication with the user. The attacker would then be free to define any job result.

STRIDE threats for the compromised user machine:

- Information disclosure. The attacker can access and inspect all sensitive assets, such as code, secrets, and dependencies.
- Tampering. The attacker can modify information sent to the worker machine, or modify files directly on the machine possibly for malicious purposes.

Chapter 4

Framework Design

A general model to CI in HREs was shown in section 3.3. In this section, a different approach to CI is presented. The proposed framework is designed for the same scope as the general CI model shown earlier. An adjusted threat model will be presented in section 4.7 along with the security requirements of the framework in section 4.9.

The resulting framework design is shown below. It is a system that attempts to protect the CI process by use of hardware isolation with Intel SGX and is designed to be compatible with an HRE.

4.1 Entities and participants

In the framework, there are the following entities and participants which can also be seen in figure 4.1.

- SGX Platform. In the context of the framework, a *worker* executes a pre-defined software in the Intel SGX environment. A *worker instance* is a short-lived instance of execution of the Intel SGX protected *worker*, which uses identical software.
- Orchestrator. The developer or user.
- Intel PCCS, which is configured to be used offline. Intel-provided software for maintaining Intel signed certificates capable of verifying SGX Platforms and SGX Quotes.

Furthermore, the framework requires participation in the form of manual administration of the PCCS.

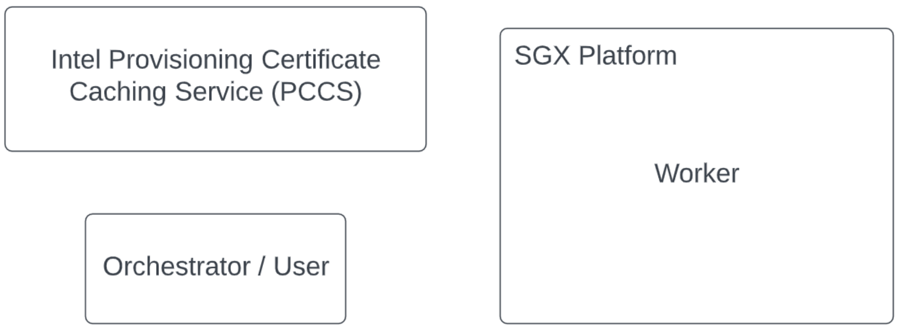

Figure 4.1: Participants in the CI framework

4.2 Overview

The framework design is intended to allow the orchestrator to define arbitrary work to be done and let a worker instance produce results verifiably. This trust is rooted in the guarantees provided by the Intel SGX hardware. This is achieved using protocols, services, and software provided by Intel and contributions from previous research. The framework design relies on attestation of the worker instance and secure communication between entities (see section 4.3), the configuration of the worker instance (see section 4.4), and verification of the produced result (see section 4.5).

4.3 Attestation and communication

Attestation and communication are provided by existing Intel protocols and through other contributions such as RA-TLS and Gramine. The measurement (which includes a hash of the memory used to run the application, flags used, etc.) for a legitimate worker instance must be known beforehand. RA-TLS is used to set up a secure communication channel between the orchestrator and SGX Platform application. RA-TLS incorporates the attestation protocol of ECDSA-attestation into the creation of the TLS certificate stored in the SGX Platform application.

The following is a high-level description of the protocol used for the initial setup that is required when an application is first launched in the SGX Platform, as can be seen in figure 4.2. Initial setup procedure:

0. The PCCS is manually configured and updated with certificates provided by PCS.

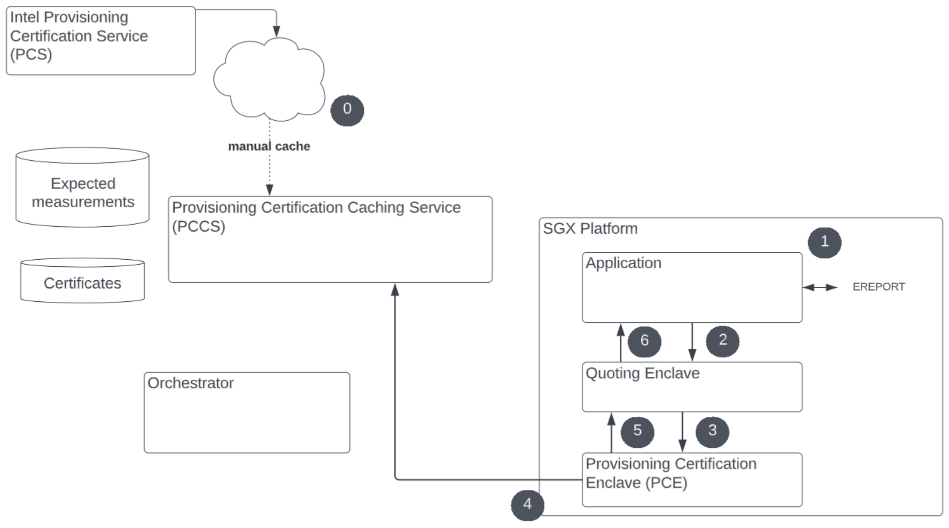
Intel Provisioning Certification Service (PCS)
0
manual cache
Expected measurements
Certificates
Provisioning Certification Caching Service (PCCS)
SGX Platform
1
Application
EREPORT
6
2
Quoting Enclave
5
3
Orchestrator
Provisioning Certification Enclave (PCE)
4

Figure 4.2: Protocol to generate a self-signed X.509 certificate in the SGX Platform application containing the SGX Quote

1. The application generates a key-pair that will be used as part of the RA-TLS protocol. The application then creates an SGX Report through the hardware instruction EREPORT. The SGX Report will include a hash of the public key of the generated key-pair.
2. The application sends the SGX Report to the Quoting Enclave.
3. The Quoting Enclave sends a public key of its attestation key to the PCE.
4. The PCE receives an attestation collateral with certificates and revocation lists from the PCCS and signs the public key of the Quoting Enclave.
5. The Quoting Enclave produces an SGX Quote signed by its attestation key.
6. The application receives the SGX Quote and produces a self-signed X.509 TLS certificate as used in RA-TLS containing the SGX Quote. The SGX Quote contains a hash of the public key used for the self-signed TLS certificate, which binds it to the enclave.

In the above procedure, step 4 is only necessary for the first generation of an SGX Quote in the SGX Platform.

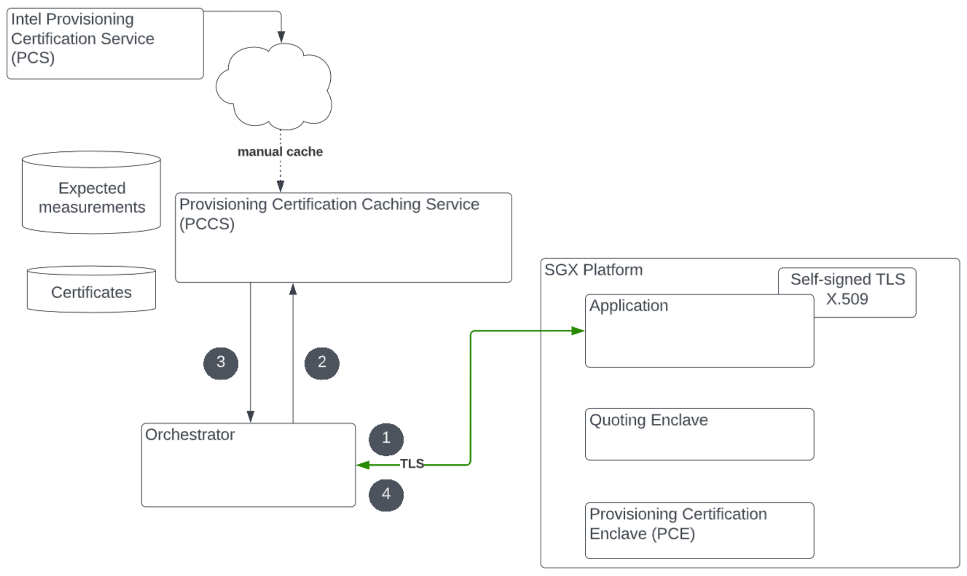
Intel Provisioning Certification Service (PCS)
manual cache
Expected measurements
Certificates
Provisioning Certification Caching Service (PCCS)
SGX Platform
Self-signed TLS X.509
Application
3
2
Quoting Enclave
Orchestrator
1
TLS
4
Provisioning Certification Enclave (PCE)

Figure 4.3: Protocol to establish a secure communication channel between the SGX Platform application and the orchestrator and verify the SGX Quote

The following is a high-level description of the establishment of a secure communication channel between the orchestrator and the SGX Platform and the verification of the SGX Quote, as can be seen in figure 4.3. Establishment procedure:

1. A TLS handshake procedure is initiated by the orchestrator to the SGX Platform application. The SGX Platform application sends the self-signed X.509 certificate containing the SGX Quote.

2. The orchestrator requests the attestation collateral for the provisioned keys of the SGX Platform that issued the Quote.

3. The orchestrator verifies that the SGX Quote is authentic and correct.

4. The TLS handshake is completed. A secure communication channel can be used.

To verify the SGX Quote (step 3), the orchestrator must compare the measurements of the application to the expected values and validate that the provisioned keys that issued the attestation key is recognized by Intel. It must also ensure that the hash of the public key associated with the TLS certificate matches that in the SGX Quote.

4.4 Configuration

When attestation has been performed and a secure communication channel has been set up. The orchestrator sends a configuration detailing the work that the worker instance should do. The *config* data object contains the following data:

- Inputs. A list of inputs, such as files and zipped folders.
- Scripts. A list of scripts that the worker should execute.
- Expected outputs. A list of expected output names. It is up to the provided scripts to store the outputs in a designated folder.

The worker instance executes the scripts in the configuration.

4.5 Verification

The worker instance generates a self-signed certificate, namely a *signing certificate*, containing the SGX Quote and hardware information similar to the TLS certificate in RA-TLS. Specifically, the *signing certificate* and the corresponding asymmetric key-pair must be bound to the SGX Quote to allow users to trust that the signer is the intended worker instance. Step 1 of the setup procedure (in section 4.3) is modified to also include the public key of the key-pair associated with the *signing certificate* in the SGX Report. Then, the public key will be included in the SGX Quote, and the SGX Quote is included in the *signing certificate*. The public key is included in the SGX Quote in full, unlike the public key associated with the TLS certificate.

The private key associated with the *signing certificate* will be used for signing results in the worker instance. The public key, associated with this certificate, along with the certificate itself, will be transmitted over the secure communication channel to the orchestrator.

The data returned to the orchestrator from the worker instance is a signed *result* data object. The *result* data object consists of a hash of the configuration that was used and a list of outputs. The *result* is signed by the worker instance with the private key associated with the *signing certificate* producing a *signed result*.

The orchestrator may verify the signature with the public key and the associated *signing certificate*. The orchestrator may need to verify the SGX Quote included in the *signing certificate* to trust the signature. If the orchestrator has attested the worker instance previously, when establishing the secure communication channel, then the SGX Quote in the *signing certificate* is identical and should not need further validation.

The worker instance is not involved in verifying the results produced. For clarification, to verify the *signing certificate*, the user needs access to the CRLs, which are accessible through PCCS in the offline network. The CRLs are associated with the provisioned keys that are used in the SGX Platform.

With access to the specific CRLs, that are trusted to be issued by Intel, and the produced *signed result*, a user can gain confidence that the intended worker instance has produced the results.

It was decided to use a separate *signing certificate* to sign the results of the worker instance, rather than use the same certificate and key-pair used for communication by RA-TLS because the key-pairs are used for different purposes.

4.6 Authenticity and trust terminology

This section defines the meaning of authenticity and trust in the context of the framework design. Specifically, an entity, such as an SGX Quote or a signing certificate, is often referred to as either authentic or trusted for the rest of the report.

Authenticity

An authentic entity is unmodified, not tampered with, and produced in an authentic environment.

Trust

A trusted entity appears authentic to participants in the framework, but it does not have to be authentic.

For example, an SGX Quote may be trusted in the framework by the orchestrator after local validation, but it does not have to be authentic. An authentic SGX Quote or signing certificate could not be trusted by any participants in the framework if, for instance, local verification is tampered with by an attacker.

4.7 Adjusted threat model

In section 3.3, a general approach to CI in HREs is presented. The framework design introduces new definitions of entities and more participants and protocols that should be considered. This section will expand on the previous model. For each participant in the framework, STRIDE threats have been identified. Additionally, threats T1-T9 to the framework are defined from the identified STRIDE threats.

4.7.1 Attacker capabilities

Mainly, two forms of attacker capabilities are considered for the SGX Platform and the worker instance. The ability to *imitate* a worker instance, and the ability to launch an *illegitimate* worker instance that appears authentic or trusted. For example, to *imitate* a worker instance, the attacker may learn internal SGX keys of the PCE, or secrets in the worker instance. To launch an *illegitimate* worker instance, the SGX Platform itself can be malicious, the execution of the worker instance can be tempered with, or expected measurements can be modified.

Furthermore, the attacker is assumed to be capable of observing and modifying actions taken by users in the framework, such as the orchestrator.

4.7.2 PCCS

The following STRIDE threats are considered for attackers with elevated privileges on the machine running PCCS:

- Tampering - Data used as part of the application, such as Intel-issued certificates, and responses from the applications may be modified.
- Information Disclosure - Information about the use of the applications and data associated with them, such as Intel-issued certificates, can be disclosed to the attacker.

Because the system is offline, the PCCS can not verify the integrity of Intel-issued certificates by itself. Modification of such files, which is assumed to be feasible on a compromised machine, could be non-trivial to detect.

Additionally, if Intel delivers malicious signed certificates. It would allow malicious SGX Platforms to appear authentic.

Accordingly, the following threats are defined for the PCCS:

T1 *Illegitimate worker instance by tampering* - Modifying certificates or measurement data held by the PCCS will allow an illegitimate worker to produce a seemingly authentic SGX Quote, hence users would successfully attest an illegitimate worker instance

T2 *Illegitimate worker instance by compromised third-party* - The attacker can compromise Intel, which acts as a trusted CA, and the signed certificate that is received by PCCS can authenticate malicious SGX Platforms in the framework.

T3 *Illegitimate worker instance by internal sabotage* - A trusted IT administrator uploads false measurement data and/or certificates that allow illegitimate worker instances.

4.7.3 Orchestrator

The following STRIDE threats are considered for a compromised orchestrator machine:

- Tampering - An attacker can modify the configuration sent to the worker instance.
- Tampering - An attacker could interfere in the verification process of the SGX Quote and make a forged SGX Quote appear authentic.
- Information Disclosure - An attacker can observe data sent to and from the worker instance.

The following threats are defined for the orchestrator:

T4 *Misconfiguration by tampering* - An attacker can modify the configuration sent to the worker instance.

T5 *Mistrust by tampering* - An attacker can interfere in the verification and induce trust in a forged SGX Quote.

T6 *Observability* - An attacker can observe the data sent, such as code and artifacts, to and from the worker instance.

4.7.4 Worker

A compromised worker machine can either run a legitimate worker instance (i.e. not modified), an illegitimate worker instance (i.e. modified), or attempt to imitate the behavior of a worker instance.

If a compromised worker machine runs a legitimate worker machine it may be possible for an attacker to:

- Information Disclosure - Leak the private key, that is associated with the *signing certificate*, that can be used to sign arbitrary results that appear authentic.
- Tampering - Modify execution of the enclave.

Running an illegitimate worker may allow the attacker to produce arbitrary results. The attacker would be required to forge measurement data that is included in the SGX Quote, for the illegitimate worker to appear as legitimate.

To imitate a legitimate worker, one possible attack threat is to gain access to the provisioned keys used as part of the PCE, which would be used to imitate a QE to sign a forged SGX Quote which would be trusted by the orchestrator.

T7 *Imitate worker instance by breaking confidentiality of PCE* - An attacker may break the confidentiality of the PCE, which could allow the attacker to gain possession of the provisioned keys. With the provisioned keys, the attacker is assumed to be able to imitate the execution of an SGX Platform and produce trusted SGX Quotes.

T8 *Imitate worker instance by breaking confidentiality of worker instance* - An attacker may break the confidentiality guarantees of the worker instance. If the attacker gains possession of the signing certificate and associated key-pair, the attacker may sign arbitrary results as if they were produced authentically.

T9 *Illegitimate worker instance by breaking integrity of the SGX Platform* - An attacker may break the integrity of the SGX Platform, hence tampering with the execution within the enclaves of the worker instance, PCE, or Quoting Enclave to make the worker instance produce modified results.

4.8 Assumptions

The following are assumptions of what an attacker is unable to do and assumes trust in certain guarantees provided by the provider of certain applications that are used.

A1 An attacker is unable to break confidentiality and integrity guarantees of Intel SGX environments. The attacker is incapable of extracting information used internally in the SGX Platform and in the various enclaves running within. It is therefore not possible for an attacker to gain access to sensitive cryptographic keys within the SGX Platform, for instance.

A2 An authentic SGX Quote includes correct and honest measurements for the application. If an illegitimate worker instance is deployed, its measurements must be different from that of a legitimate instance. Furthermore, an attacker cannot forge measurements of a legitimate instance and, hence, produce authentic SGX Quotes.

A3 Intel's general certificate authority, which is used to validate the provisioned keys in each SGX Platform, is assumed to be trusted. An attacker is assumed not to be able to generate false certificates as issued by Intel.

A4 A legitimate worker instance is assumed not to leak sensitive information by accident through communications with other entities or by other means. Furthermore, the worker software, in a legitimate worker instance, is assumed to act honestly and produce a result as expected from the configuration.

4.9 Security requirements

The security requirements below aim to mitigate the threats to the Worker. The requirements aim to ensure that a signed *result* data object is authentic only if it is produced in a legitimate worker instance. The integrity and authenticity properties of the produced objects are prioritized.

S1 The *signing certificate* is authentic if and only if the SGX Quote is authentic.

S2 An authentic SGX Quote can only be produced for a legitimate worker instance in a known platform.

S3 A legitimate worker instance will produce an authentic *result* data object from the *config* data object.

S4 A signing certificate in a legitimate worker instance cannot be leaked.

Not all threats in section 4.7 are mitigated. The framework is not designed to mitigate threats to the orchestrator and PCCS. Specifically, it will not preserve the confidentiality of the configuration (i.e., code, files, scripts, and secrets) in the framework, it will not prevent an attacker to submit a malicious configuration to the worker instance, nor will it prevent an unauthentic SGX Quote to appear authentic to the Orchestrator either through tampering with the PCCS or the orchestrator itself.

4.10 Functional requirements

- Offline - The framework should be operable in an offline environment.
- Customization - The framework should allow arbitrary applications to be built and executed.
- Longevity of verification – The verification process of the result must be independent of the worker instance (i.e. the worker instance can be terminated), and the signed result can be verified long after its creation and by third parties.
- Limited Overhead - The framework should not introduce significant overhead.

Chapter 5

Results and analysis

This section presents the assurance case developed for the framework in chapter 4. The goal is to show that the security requirements are fulfilled given the assumptions and threats defined for the proposed framework.

5.1 Goals and evidence

In the safety case, each security requirement in section 4.8, namely S1-S4, is mapped to goals G1, G4, G9, and G12 respectively. The Goal Structuring Notation is presented in section 5.1.1. Each principal element in the GSN is further described below. Furthermore, evidence is provided for each goal individually in section 5.1.2.

5.1.1 Goals

The Goal Structuring Notation for each security requirement is presented in this section. All goals and sub-goals make up G1-G12. There are three strategies, St1-St3, and there are three contexts, C1-C3.

5.1.1.1 GSN of S1

Figure 5.1 presents the notation of the goal G1 - “The signing certificate is authentic if and only if the SGX Quote is authentic”. The strategy, St1, ensures that the authenticity of the signing certificate is at least identical to the authenticity of the SGX Quote. For clarification, the authenticity property of the signing certificate and the SGX Quote is deterministic, either it is authentic or it is not. In the end, goals G2 and G3 aim to fulfill the equivalence nature of goal G1.

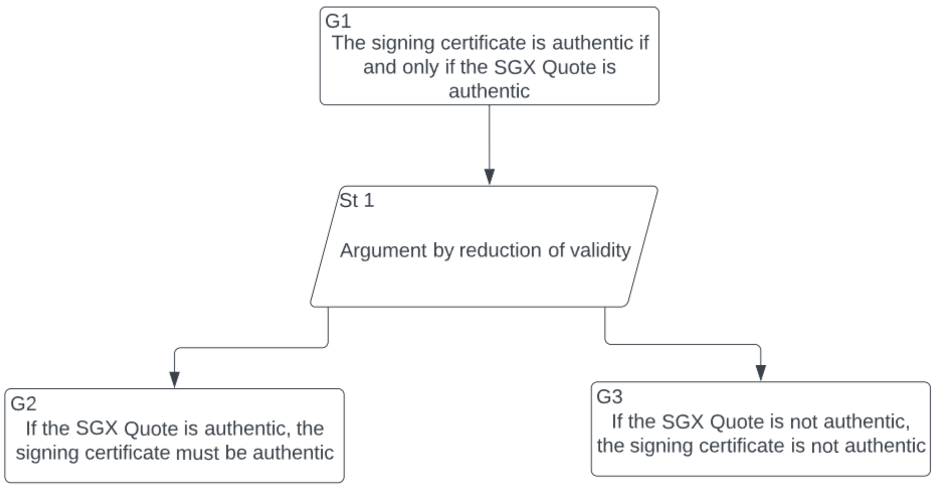
G1
The signing certificate is authentic if and only if the SGX Quote is authentic
St 1
Argument by reduction of validity
G2
If the SGX Quote is authentic, the signing certificate must be authentic
G3
If the SGX Quote is not authentic, the signing certificate is not authentic

Figure 5.1: GSN of security requirement S1

5.1.1.2 GSN of S2

Figure 5.2 presents the notation of the goal G4 - “An authentic SGX Quote can only be produced by a legitimate worker in a recognized SGX Platform”. The first strategy, St2, ensures that an authentic SGX Quote can be produced as described, establishing the base case in this context. The second strategy, St3, ensures that no other entity can produce an authentic SGX Quote under other conditions and takes a vulnerability-based approach. An exhaustive approach to St3 is not feasible, therefore, context C1 limits the argument’s scope to only known attack threats on the Worker as mentioned in section 4.7.

Specifically, all known attack threats relating to goal G4 are T3, T7, and T9. The context does not consider threats T1-T2 because they are directed at the PCCS service. Threats T1-T2 will be revisited later. G7 concerns threats T3 and T9, which could allow an attacker to influence how the SGX Quote is produced. G8 concerns threat T7, which could allow an attacker to imitate a worker instance by being able to produce an SGX Quote that is trusted in the network.

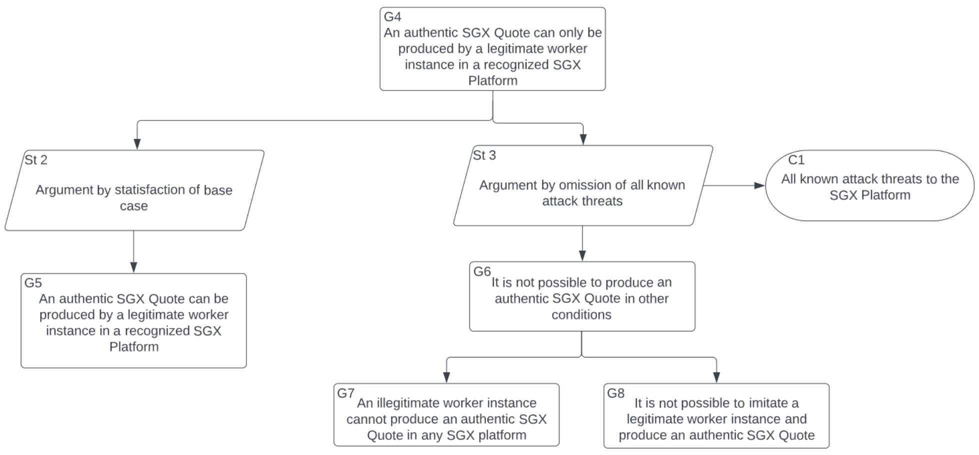
G4
An authentic SGX Quote can only be produced by a legitimate worker instance in a recognized SGX Platform
St 2
Argument by statisfaction of base case
St 3
Argument by omission of all known attack threats
C1
All known attack threats to the SGX Platform
G5
An authentic SGX Quote can be produced by a legitimate worker instance in a recognized SGX Platform
G6
It is not possible to produce an authentic SGX Quote in other conditions
G7
An illegitimate worker instance cannot produce an authentic SGX Quote in any SGX platform
G8
It is not possible to imitate a legitimate worker instance and produce an authentic SGX Quote

Figure 5.2: GSN of security requirement S2

5.1.1.3 GSN of S3

Figure 5.3 presents the notation of the goal G9 - "A legitimate worker instance will produce an honest result data object from the config data object". Two sub-goals are identified. The first goal, G10, ensures that the SGX Platform can ensure the integrity of the executing software. Goal G10 is in response to threat T9, which is the only identified threat to G9 that is a part of the threat model. Goal G11 is an undeveloped goal but ensures that the worker software executes without unintended errors and that worker software will produce a result. The worker software itself is undeveloped, therefore this goal was also decided to be undeveloped.

Context C2 is associated with goal G9, which specifies that threat T9 from the threat model, and additionally, threats concerning the implementation of the worker software, are the only known threats to fulfill G9. Threats in context C3 are mitigated in goals G10 and G11.

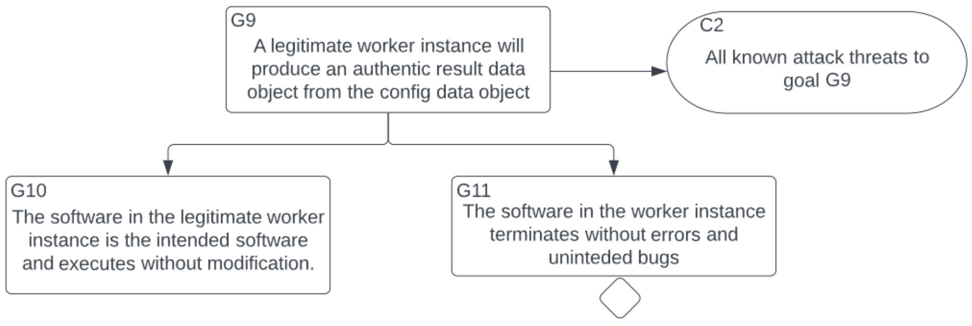
G9
A legitimate worker instance will produce an authentic result data object from the config data object
C2
All known attack threats to goal G9
G10
The software in the legitimate worker instance is the intended software and executes without modification.
G11
The software in the worker instance terminates without errors and uninteded bugs

Figure 5.3: GSN of security requirement S3

5.1.1.4 GSN of S4

Figure 5.4 presents the notation of goal G12 - "The private key associated with the signing certificate in a legitimate worker instance cannot be leaked", which is identical to security requirement S4. Context C3, all known attack threats to the worker instance, is attached to G12.

Specifically, only threat T8 is included in context C3, as it is the only known threat to the worker instance.

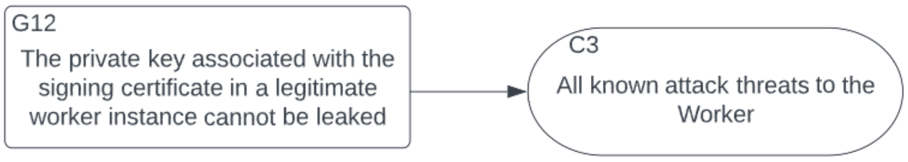
G12
The private key associated with the signing certificate in a legitimate worker instance cannot be leaked
C3
All known attack threats to the Worker

Figure 5.4: GSN of security requirement S4

5.1.2 Evidence

This section will first present each goal that does not have additional sub-goals attached and the evidence that aims to sufficiently fulfill them. G11 is an undeveloped goal, and will not be included. Evidence will be in the form of an argument, which is based on assumptions or other premises relating to the framework.

Goal G2 - "If the SGX Quote is authentic, the signing certificate must be authentic"

The SGX Platform has confidentiality and integrity guarantees by assumption A1. An authentic SGX Quote cannot have forged measurements, by assumption A2, so the application must be the intended one. A hash of the public key associated with the signing certificate included in the SGX Quote ensures that the creator of the signing certificate must also generate the SGX Quote. The sensitive keys cannot be extracted from the application. The application that created the signing certificate must therefore be the same application that requested the SGX Quote. An authentic SGX Quote, therefore, indicates that the application is the intended application and that the signing certificate, therefore, is authentic.

Goal G3 - "If the SGX Quote is not authentic, the signing certificate is not authentic"

If the SGX Quote is not authentic, by the procedure of the verification of a signing certificate, the signing certificate must also not be authentic.

Goal G5 - "An authentic SGX Quote can be produced by a legitimate worker instance in a recognized SGX Platform"
By design in the Intel SGX framework, a legitimate worker instance can produce an SGX Quote, which is verifiable to be authentic.

Goal G7 - "An illegitimate worker instance cannot produce an authentic SGX Quote in any SGX Platform"
Threats T3 and T9 are considered under context C1 and allow an illegitimate worker instance to appear authentic. For threat T9, by assumption A2, an attacker is incapable of forging measurements for a worker instance. By assumption A1, attackers cannot modify execution or data in the SGX Platform, which guarantees the integrity and authenticity of the SGX Quote.

Furthermore, relating to threat T3, by assumption A3, an illegitimate instance cannot appear authentic and hence produce an authentic SGX Quote by compromising Intel and forging certificates for malicious SGX Platforms.

Goal G8 - "It is not possible to imitate a legitimate worker instance and produce an authentic SGX Quote"
By assumption A1, confidentiality and integrity of the SGX Platform are guaranteed. In the context of threat T7, an attacker requires access to the keys used internally in the SGX Platform by either the Quoting Enclave or the PCE to be able to imitate a legitimate worker instance and produce an authentic SGX Quote.

Goal G10 - "The software in the legitimate worker instance is the intended software and executes without modification."
A legitimate worker instance has the intended measurements, by definition. By assumption A2, it must be the intended software. By assumption A1, the integrity of the application is preserved and the execution order is guaranteed.

Goal G12 - "The private key associated with the signing certificate in a legitimate worker instance cannot be leaked"
By assumption A1, the confidentiality of the worker instance holds in a legitimate worker instance. Furthermore, by assumption A4, the

private key cannot leak through unintended human errors in the form of software bugs.

Next, evidence is presented for the remaining goals, which depend on the goals above.

Goal G1 - "The signing certificate is authentic if and only if the SGX Quote is authentic"
By strategy St1, the authenticity equivalence of the signing certificate and the SGX Quote (G1) is logically decomposed into G2 and G3, which both must be true for the equivalence to hold. From the above, G2 and G3 hold.

Goal G4 - "An authentic SGX Quote can only be produced by a legitimate worker instance in a recognized SGX Platform"
By induction, St2 and St3 satisfy the base case and the inductive step, respectively. The base case is satisfied by goal G5. The sufficiency of satisfaction of the inductive step in goal G6 is limited by context C1, which only considers known attack threats as mentioned in section 5.1.1.2.

Goal G6 - "It is not possible to produce an authentic SGX Quote in other conditions"
The goal is limited by context C1, which restricts the goal to all known attack threats on the authenticity of the SGX Quote. Each known attack threat is satisfied in G7 and G8.

Goal G9 - "A legitimate worker instance will produce an authentic result data object from the config data object."
The goal is decomposed into G10 and G11. G10 is fulfilled, as mentioned above. G11 is outside the scope of this work. G9 is further fulfilled by assumption A4, namely that the worker instance acts honestly if the enclave has not been tampered with.

5.2 Analysis

The validity and soundness of the claims associated with each goal above are evaluated in this section. The validity is often based trivially on the premises. In section 5.2.3, the validity and soundness are further evaluated to establish a motivation for the level of sufficiency of fulfilling the goals G1-G12 and security requirements S1-S4.

5.2.1 Validity

Goal	Premises	Context
G2	A1, A2	-
G3	-	-
G5	-	-
G7	A1, A2, A3	-
G8	A1	-
G10	A1, A2	-
G12	A1, A4	C3
G1	G2, G3	-
G4	G5, G6	-
G6	G7, G8	C1
G9	G10, G11	C2

Table 5.1: Goals' premises and contexts of the assurance case

The relationship between the goals, premises, and contexts is further clarified in table 5.1. The relationship between the elements is used to analyze the validity. Goal G2 is trivially fulfilled based on premises A1 and A2. Goal G3 follows the definition of authenticity and the signing certificate. Goals G5, G7, G8, and G10 are trivially fulfilled based on their premises.

Goals G1, G4, G9, and G12 are the top-level goals that are also the security requirements S1-S4. To analyze the validity of argumentation for these goals, respect must be taken to the chosen strategies associated with them, and how the sub-goals are combined to fulfill the top-level goals.

To fulfill G1, strategy St1 is used, which creates sub-goals that fulfill the equivalence nature of goal G1. No context is associated with the strategy or the goals. Because the sub-goals G2 and G3 are themselves valid, the validity of G1 is ensured.

To fulfill G4, an inductive approach is used with strategies St2, for the base case, and St3, for the inductive step. Goal G5, associated with the base case of the inductive approach, is trivially valid. Context C1 is associated with the inductive step, which significantly limits the scope of the argument. However, under context C1, goal G6 is valid because each threat in context C1 is mitigated in the framework. This propagates to goal G4, which is also valid under context C1.

To fulfill G9, no explicit strategy is used. By definition, goal G9 concerns how a legitimate worker instance can produce an honest result. Threats in context C2, which are mitigated by goals G10 and G11, are sufficient to ensure the validity of G9. There may be potential threats that are overlooked in this work. However, under context C2, G9 is valid.

To fulfill G12, the context C3 is considered. The validity of G12 is ensured under context C2.

5.2.2 Soundness

Assumption	Confidence
A1	Low
A2	Low
A3	Low*
A4	-

Table 5.2: The confidence level associated with each assumption.

The soundness of the arguments depends on if the premises, which are the assumptions A1-A4, are true. This section will therefore dissect the assumptions based on findings in the literature and especially the threat model presented in section 2.5.4.

The assumptions made for the framework are not all mutually independent, which means there may be an overlap in specific weaknesses. As seen in table 5.2, the confidence level, indicating how true an assumption is likely to be, is associated with each assumption. The confidence is one of *Low*, *Mid*, and *High*. However, no assumption is assigned a *Mid* or *High* confidence level in this work. The confidence levels are motivated below.

Assumption A1 is about the confidentiality and integrity guarantees in the SGX Platform. The threat model by Costan et al. [12] presented in section 2.5.4 highlights that the protection in Intel SGX does not hold up to sophisticated attacks. In this framework, the attacker is assumed to be highly sophisticated. The attacker could make use of cache-timing attacks that would not require privileged access to the computer running the worker instance. Consequentially, the attacker could break confidentiality and possibly extract keys used for signing the outputs of the worker instance, which would completely break the framework security requirements. Intel SGX also receives criticism because it is unclear to security researchers how some of the protection mechanisms work as they are undocumented.

Furthermore, Intel SGX is not secure against most hardware probing attacks such as chip attacks and power analysis attacks, which would also break assumption A1. The framework is intended for a Highly Regulated

Environment, where any access to the entities is more difficult; however, this work assumes that attackers have access to the entities in the framework, as mentioned in section 4.7.1. The confidence level for assumption A1 is *Low*.

Assumption A2 relates to assumption A1, because an immediate effect of breaking integrity can be to break A2. The assumption states that the measurements in an SGX Quote of an illegitimate worker instance must differ from the measurement in a legitimate worker instance. The confidence level for assumptions A2 is therefore also *Low*.

Assumption A3 assumes trust in Intel as a certificate authority to identify authentic SGX platforms. Such an attack is difficult but potentially feasible for a persistent and capable attacker. There may also be other risks not considered in this work regarding this assumption. The confidence level associated with the assumption is *Low*, as it is not within the scope of this work to evaluate this risk.

Assumption A4 assumes that there are no unintended leaks in the form of bugs or misconfiguration of the Intel SGX Platform or the Worker software, which is outside the scope of this work. Such errors could compromise the secrets used in the worker instances. No confidence level is associated with this assumption, as there is no implementation of the worker software, and its related goal G12 is also undeveloped in the safety case.

5.2.3 Security analysis

In this section, the security requirements are analyzed in relation to the validity and soundness of arguments that aim to fulfill them. The sufficiency of argumentation is analyzed in section 5.2.3.1. In section 5.2.3.2, it is explained to what extent threats are mitigated in the proposed framework.

5.2.3.1 Sufficiency of argument

A low confidence level is associated with assumptions A1 and A2. Goals G2, G7, G8, G10, and G12 directly depend on these assumptions. Concerning soundness, the weaker confidence level thus propagates to each security requirement S1-S4.

Validity is primarily constrained by contexts C1-C3, which limits the threats that are considered to what is known in this work. Validity is however justifiable with respect to the given contexts.

The attacker is assumed to be persistent and highly capable; therefore, the sufficiency of argument can be argued not to be sufficient in relation to the considered threat model, where the goal is to ensure that security requirements

S1-S4 always hold. Increased soundness would significantly increase the sufficiency of argumentation.

5.2.3.2 Mitigated threats

Threats T3 and T7-T9 are mitigated by goals G6, G9, and G12 through contexts C1-C3. The remaining threats will be discussed in this section.

Threats T1 and T2 make changes to which SGX Platforms are identified as legitimate in the framework. An illegitimate worker instance may thus appear legitimate. However, by security requirement S1, the signing certificate is equally authentic as the SGX Quote. Furthermore, an illegitimate worker instance cannot produce an SGX Quote for a different SGX Platform. A result data object must include a signature from the signing certificate, consequentially, it would be possible to distinguish the results that were produced by the false legitimate SGX Platform by a user that knew that the certificate was false.

Threats T4, T5, and T6 involve making changes to how the orchestrator, or user, communicates the configuration or verifies the result of the worker instance. With security requirements S1-S4 and how the result is produced, the result data object must reflect changes in the configuration or fail verification if it has been tampered with. However, participants in the framework may not be able to verify this property.

Threats T1, T2, and T4-T6 have in common that participants in the framework are vulnerable to the threats, but if verification of the result is performed in an honest environment, possibly outside the framework, it can be known to be authentic.

Chapter 6

Discussion

6.1 Guarantees and trade-offs

The security requirements were chosen to guarantee certain properties in the CI process. Specifically, the integrity of resources if they originate from a worker instance and that the authenticity of the resources can be verified. The confidentiality of resources is not protected in the framework. This means that an attacker may still learn details about the resources as they are transferred between participants in the network, despite that such resources could be sensitive. More importantly in this work, an attacker is unable to modify resources, as it would break the authenticity and integrity properties.

Integrity and authenticity were prioritized over confidentiality, because of the characteristics of a supply chain attack. Which is the problem this framework addresses.

The usability of the framework is a concern, as it does not solve any problems surrounding the SDLC in HREs; instead, it introduces additional complications. Parity between environments was previously a problem, and building and testing the software in hardware isolation has the risk of creating more parity, which could introduce security risks.

The framework's role is to, in the end, mitigate supply chain attacks. The verification of the resources created by a worker instance, such as the software product, is not limited to being performed within the framework. This means that there could be multiple lines of defense to ensure that the produced software is correct.

In the case of the SolarWinds attack, the insertion of malicious code or the modification of binaries should be visible if either the *result* data object fails verification, or if the *result* data object does not reflect the intended *config* data object. The *config* data object can be verified to have contained the intended files, and the software produced can be known to be authentic from the valid *result* data object.

6.2 Threats to validity

From this work alone, it cannot be determined if the framework is implementable as it is defined. Furthermore, the extent to which arbitrary execution of software can be employed in Intel SGX Platforms is unclear. Tools such as Gramine have exemplified that similar use cases are possible. However, it cannot be determined for this framework's use cases, such as building and testing arbitrary applications, and exactly what limitations that implies. A proof of concept is missing from this work, which is important for the practicality and thus the validity of the theoretical framework.

The research method used in this work is qualitative. A safety case is used to argue for the level of security and its argumentation's sufficiency. A more conclusive result could have been the product of a more extensive research method. A more extensive research method should have more rigorous arguments and possibly proofs, by formal methods, for the requirements of the framework to hold.

Furthermore, a relaxed set of assumptions is important to complement this work. The safety case is highly sensitive to changes in assumptions, if an assumption would be false, the validity of the argumentation is compromised.

6.3 Shift of trust

A characteristic of HREs is that they use air-gapped environments, primarily to eliminate the trust that needs to be put in routers, firewalls, and configurations to protect the resources in the network. This framework takes it one step further and attempts to eliminate the trust one needs to put in the participants themselves, such that no computer or user is assumed to be trusted. Instead, the trust is placed in Intel and the solutions they provide.

When attempting to secure the CI process in the context of the supply chain, it is difficult to predict an attacker's capabilities and potential threats. Building software in an HRE decreases this cyber risk and many potential threats are immediately mitigated; however, there is still considerable cyber risk. For example, the internal threat of an employee introducing malicious software into the network, whether it is accidental or deliberate, is difficult to mitigate.

Placing trust in Intel to guarantee the security properties of the CI process may be misplaced. However, it could be argued that the attack surface and potential threats are more predictable because the threats apply to Intel's hardware isolation technology rather than a physical air-gapped network with many actors.

6.4 Improved hardware isolation

The confidence that Intel SGX can fulfill the security requirements for the framework in this work is low, mainly because of the previously found weaknesses in Intel SGX. Selecting other hardware isolation technologies may allow increased confidence in the framework's security requirements. Unless changes are made to Intel SGX, other hardware isolation technologies such as Sanctum or ARM TrustZone could be preferable for the proposed framework.

Sanctum is designed to withstand more sophisticated attacks, and the potential to use Sanctum for the proposed framework should be evaluated further in future work. However, implementing the framework with Sanctum could be difficult in its current state. Because to the best of the author's knowledge, it does not offer the same degree of developer tools, such as SDKs, libraries, and other open-source contributions compared to Intel SGX to make it easier or perhaps feasible.

The proposed framework isolates all data and operations. A semi-sandboxed variant, where only the essential data and operations are protected, may be better suited if another hardware isolation technology is used in the framework. Identifying which data and operations can be separated without sacrificing security is suggested for future work. A semi-sandboxed variant could also address usability concerns regarding the performance of the proposed framework.

Chapter 7

Conclusions and future work

7.1 Conclusions

With the used security context (assumptions, requirements, and threat model), a hardware isolation approach using Intel SGX has the potential to guarantee the integrity and authenticity of the CI process in HREs. However, the security weaknesses in Intel SGX do not allow confidence in the assumptions of the framework. In its current state, the security properties of the framework are not sufficiently fulfilled.

Placing trust in Intel SGX to secure the CI process rather than in existing practices in HREs could reduce the attack surface and potential threats, which could allow practitioners to make better assertions about the security of the produced resources. However, more research and progress on the guarantees of the Intel SGX technology, as well as alternative hardware isolation solutions, is necessary for this trust to be sufficiently motivated.

7.2 Limitations

The largest limiting factors are the choice of assumptions, and the scope of the threats analyzed for the framework. Furthermore, it was decided to not implement a proof-of-concept, which is necessary to evaluate the framework further and to ensure that it is possible to use the proposed technologies in the manner that is intended.

7.3 Future work

Due to the breadth of the problem, the goals have only been met to a certain extent. Below, some of the remaining issues that should be addressed in future work are presented.

- A prototype of the framework needs to be implemented as a proof-of-concept of the proposed framework.
- A more exhaustive threat model on the participants in the framework is important to strengthen the argument's sufficiency.
- The argument's validity can be strengthened by a more rigorous research method, such as by using formal methods.
- How confidentiality of data within the framework can be secured should be explored further. Currently, confidentiality is not protected.
- Choosing a different hardware isolation technology such as Sanctum could lead to significantly strengthened security properties of the proposed framework.
- A semi-sandboxed hardware isolation approach, which does not sacrifice security, should be explored further. It could address usability concerns.

7.4 Reflections

This work introduces a new approach adapted for the CI process using hardware isolation. The adoption of a new approach to CI could be costly for practitioners in HREs and could mean that organizations must make investments in establishing new regulations and requirements for software development.

The theoretical framework could have a significant impact on the security of supply chains by mitigating more threats than in an HRE alone. In this way, the approach could have significant economic and social effects if it can prevent attacks on large and vulnerable organizations. However, the advantage of the framework needs further evaluation, refinement, and implementation, which is an investment that needs to be put in comparison to potential gains in security.

References

[1] A. Dolgui, D. Ivanov, and B. Sokolov, “Ripple effect in the supply chain: an analysis and recent literature,” *International Journal of Production Research*, vol. 56, pp. 1–17, 10 2017. doi: 10.1080/00207543.2017.1387680 [Page 1.]

[2] D. A. Ghadge, M. Weib, N. Caldwell, and R. Wilding, “Managing cyber risk in supply chains: A review and research agenda,” *Supply Chain Management*, pp. 1–36, 07 2019. doi: 10.1108/SCM-10-2018-0357 [Page 1.]

[3] Institute of Risk Management, “Cyber risk - executive summary,” 2015. [Online]. Available: https://web.actuaries.ie/press/erm-resource-database/cyber-risk-executive-summary [Page 1.]

[4] A. Boone, “Cyber-security must be a c-suite priority,” *Computer Fraud & Security*, vol. 2017, pp. 13–15, 02 2017. doi: 10.1016/S1361-3723(17)30015-5 [Page 1.]

[5] European Union Agency for Cybersecurity, “Threat Landscape for Supply Chain Attacks,” https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks, 2021. [Pages 1, 15, and 16.]

[6] Kaseya, “Incident overview & technical details – kaseya.” [Online]. Available: https://helpdesk.kaseya.com/hc/en-gb/articles/4403584098961-Incident-Overview-Technical-Details [Page 1.]

[7] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017. doi: 10.1109/ACCESS.2017.2685629 [Pages xi, 2, 8, and 9.]

[8] J. A. Morales, H. Yasar, and A. Volkman, “Implementing devops practices in highly regulated environments,” in *Proceedings of*

the 19th International Conference on Agile Software Development: Companion, ser. XP '18. New York, NY, USA: Association for Computing Machinery, 2018. doi: 10.1145/3234152.3234188. ISBN 9781450364225. [Online]. Available: https://doi-org.focus.lib.kth.se/10.1145/3234152.3234188 [Pages 2 and 13.]

[9] A. Muñoz, A. Farao, J. R. C. Correia, and C. Xenakis, "P2ise: Preserving project integrity in ci/cd based on secure elements," *Information*, vol. 12, no. 9, 2021. doi: 10.3390/info12090357. [Online]. Available: https://www.mdpi.com/2078-2489/12/9/357 [Pages 2 and 24.]

[10] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2022. doi: 10.1109/MS.2021.3073045 [Page 2.]

[11] W. Zheng, Y. Wu, X. Wu, C. Feng, Y. Sui, X. Luo, and Y. Zhou, "A survey of intel sgx and its applications," *Frontiers of Computer Science*, vol. 15, no. 3, pp. 1–15, 2021. [Pages 2, 4, 16, 19, and 23.]

[12] V. Costan and S. Devadas, "Intel sgx explained," Cryptology ePrint Archive, Report 2016/086, 2016, https://ia.cr/2016/086. [Pages 3, 4, 16, 17, 18, 19, 20, 21, 22, and 50.]

[13] N. B. Ruparelia, "Software development lifecycle models," *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 3, p. 8–13, may 2010. doi: 10.1145/1764810.1764814. [Online]. Available: https://doi-org.focus.lib.kth.se/10.1145/1764810.1764814 [Page 7.]

[14] P. Rimba, L. Zhu, L. Bass, I. Kuz, and S. Reeves, "Composing patterns to construct secure systems," in *2015 11th European Dependable Computing Conference (EDCC)*, 2015. doi: 10.1109/EDCC.2015.12 pp. 213–224. [Pages 9 and 24.]

[15] R. Bloomfield and P. Bishop, "Safety and assurance cases: Past, present and possible future – an adelard perspective," in *Making Systems Safer*, C. Dale and T. Anderson, Eds. London: Springer London, 2010. ISBN 978-1-84996-086-1 pp. 51–67. [Pages 9 and 11.]

[16] S. Samonas and D. Coss, "The cia strikes back: Redefining confidentiality, integrity and availability in security." *Journal of Information System Security*, vol. 10, no. 3, 2014. [Page 10.]

[17] G. Dhillon and J. Backhouse, "Technical opinion: Information system security management in the new millennium," *Communications of The ACM - CACM*, vol. 43, pp. 125–128, 07 2000. doi: 10.1145/341852.341877 [Page 10.]

[18] R. Scandariato, K. Wuyts, and W. Joosen, "A descriptive study of microsoft's threat modeling technique," *Requirements Engineering*, vol. 20, no. 2, pp. 163–180, Jun 2015. doi: 10.1007/s00766-013-0195-2. [Online]. Available: https://doi.org/10.1007/s00766-013-0195-2 [Page 10.]

[19] J. Inge, "Defence standard 00-56 issue 4: Safety management requirements for defence systems," 2007. [Page 11.]

[20] T. Kelly and R. Weaver, "The goal structuring notation–a safety argument notation," in *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*. Citeseer, 2004, p. 6. [Pages 11 and 12.]

[21] J. A. Morales, H. Yasar, and A. Volkmann, "Weaving security into devops practices in highly regulated environments," *International Journal of Systems and Software Security and Protection (IJSSSP)*, vol. 9, no. 1, pp. 18–46, 2018. [Pages 13 and 14.]

[22] M. Guri, B. Zadov, D. Bykhovsky, and Y. Elovici, "Powerhammer: Exfiltrating data from air-gapped computers through power lines," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1879–1890, 2020. doi: 10.1109/TIFS.2019.2952257 [Pages 14, 23, and 24.]

[23] Försvarsmakten, "KSF v3.1," https://kryptera.se/assets/uploads/2017/11/must-krav-pa-godkanda-sakerhetsfunktioner-version-3-1.pdf, 2014. [Page 14.]

[24] A. Ahmad, J. Webb, K. C. Desouza, and J. Boorman, "Strategically-motivated advanced persistent threat: Definition, process, tactics and a disinformation model of counterattack," *Computers & Security*, vol. 86, pp. 402–418, 2019. doi: https://doi.org/10.1016/j.cose.2019.07.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818310988 [Page 15.]

[25] L. Lazarovitz, "Deconstructing the solarwinds breach," *Computer Fraud & Security*, vol. 2021, no. 6, pp. 17–19, 2021. doi: 10.1016/S1361-3723(21)00065-8. [Online]. Available: https://doi.org/10.1016/S1361-3723(21)00065-8 [Page 15.]

[26] S. Raponi, M. Caprolu, and R. Di Pietro, "Beyond solarwinds: The systemic risks of critical infrastructures, state of play, future directions." *ITASEC*, vol. 21, pp. 07–09, 2021. [Page 15.]

[27] J. Yang, Y. Lee, and A. P. McDonald, *SolarWinds Software Supply Chain Security: Better Protection with Enforced Policies and Technologies*. Cham: Springer International Publishing, 2022, pp. 43–58. ISBN 978-3-030-92317-4. [Online]. Available: https://doi.org/10.1007/978-3-030-92317-4_4 [Page 15.]

[28] M. A. Mukhtar, M. K. Bhatti, and G. Gogniat, "Architectures for security: A comparative analysis of hardware security features in intel sgx and arm trustzone," in *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*, 2019. doi: 10.1109/C-CODE.2019.8680982 pp. 299–304. [Pages 16, 20, 22, and 23.]

[29] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 857–874. [Pages 16 and 23.]

[30] Intel, "Design guide for intel® sgx provisioning certificate caching service (intel® sgx pccs)," https://www.download.01.org/intel-sgx/sgx-dcap/1.10/linux/docs/SGX_DCAP_Caching_Service_Design_Guide.pdf. [Pages xi, 17, 18, and 19.]

[31] Intel, "Attestation Services for Intel® Software Guard extensions," https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html. [Page 17.]

[32] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The ssl landscape: A thorough analysis of the x.509 pki using active and passive measurements," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: Association for Computing Machinery, 2011. doi: 10.1145/2068816.2068856. ISBN 9781450310130 p. 427–444. [Online]. Available: https://doi-org.focus.lib.kth.se/10.1145/2068816.2068856 [Page 18.]

[33] M. Guri, B. Zadov, D. Bykhovsky, and Y. Elovici, "Ctrl-alt-led: Leaking data from air-gapped computers via keyboard leds," in *2019*

IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), vol. 1, 2019. doi: 10.1109/COMPSAC.2019.00118 pp. 801–810. [Page 23.]

[34] V. Gruhn, C. Hannebauer, and C. John, “Security of public continuous integration services,” in *Proceedings of the 9th International Symposium on Open Collaboration*, ser. WikiSym ’13. New York, NY, USA: Association for Computing Machinery, 2013. doi: 10.1145/2491055.2491070. ISBN 9781450318525. [Online]. Available: https://doi-org.focus.lib.kth.se/10.1145/2491055.2491070 [Page 24.]

[35] Gramine, “Gramine project,” https://gramineproject.io/. [Pages 24 and 25.]

[36] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij, “Integrating remote attestation with transport layer security,” 2018. [Online]. Available: https://arxiv.org/abs/1801.05863 [Page 24.]

TRITA-EECS-EX-2023:111