Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Dissertação de Mestrado

Integrating SPIFFE and SCONE to Enable Universal Identity Support for Confidential Workloads

Matteus Sthefano Leite da Silva

Campina Grande, Paraíba, Brasil

08/2021

Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Coordenação de Pós-Graduação em Ciência da Computação

Integrating SPIFFE and SCONE to Enable Universal Identity Support for Confidential Workloads

Matteus Sthefano Leite da Silva

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Sistemas de Computação

Andrey Elísio Monteiro Brito e Francisco Vilar Brasileiro

(Orientador e Coorientador)

Campina Grande, Paraíba, Brasil

S586i Silva, Matteus Sthefano Leite da.

Integrating SPIFFE and SCONE to enable universal identity support for confidential workloads / Matteus Sthefano Leite da Silva. – Campina Grande, 2021.

76 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2021.

"Orientação: Prof. Dr. Andrey Elísio Monteiro Brito, Prof. Dr. Francisco Vilar Brasileiro".

Referências.

1. Computação – Segurança. 2. Provisionamento de Identidades. 3. Computação Confidencial. 4. Intel SGX. I. Brito, Andrey Elísio Monteiro. II. Brasileiro, Francisco Vilar. III. Título.

CDU 004.056(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA SEVERINA SUELI DA SILVA OLIVEIRA CRB-15/225

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
POS-GRADUACAO CIENCIAS DA COMPUTACAO
Rua Aprigio Veloso, 882, - Bairro Universitario, Campina Grande/PB, CEP 58429-900

FOLHA DE ASSINATURA PARA TESES E DISSERTAÇÕES

MATTEUS STHEFANO LEITE DA SILVA

INTEGRATING SPIFFE AND SCONE TO ENABLE UNIVERSAL IDENTITY SUPPORT FOR CONFIDENTIAL WORKLOADS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação como pré-requisito para obtenção do título de Mestre em Ciência da Computação.

Aprovada em: 26/08/2021

Prof. Dr. ANDREY ELÍSIO MONTEIRO BRITO - Orientador - UFCG

Prof. Dr. FRANCISCO VILAR BRASILEIRO - Orientador - UFCG

Prof. Dr. REINALDO CÉZAR DE MORAIS GOMES - Examinador Interno - UFCG

Prof. Dr. EDUARDO DE LUCENA FALCÃO,- Examinador Externo - UFRN

Profa. Dra. MICHELLE SILVA WANGHAM - Examinadora Externa - UNIVALI

Documento assinado eletronicamente por **Eduardo de Lucena Falcão**, **Usuário Externo**, em 26/08/2021, às 18:57, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **REINALDO CEZAR DE MORAIS GOMES**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 26/08/2021, às 20:41, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **FRANCISCO VILAR BRASILEIRO**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 26/08/2021, às 20:48, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

Documento assinado eletronicamente por **ANDREY ELISIO MONTEIRO BRITO**, **PROFESSOR(A) DO MAGISTERIO SUPERIOR**, em 26/08/2021, às 21:09, conforme horário oficial de Brasília, com fundamento no art. 8º, caput, da Portaria SEI nº 002, de 25 de outubro de 2018.

A autenticidade deste documento pode ser conferida no site https://sei.ufcg.edu.br/autenticidade, informando o código verificador **1724468** e o código CRC **D99C3A0F**.

Referência: Processo nº 23096.050899/2021-28 SEI nº 1724468

Resumo

Softwares modernos não executam mais em conjuntos isolados de máquinas protegidas por estratégias e ferramentas de segurança de perímetro. À medida que empresas movem suas cargas de trabalho para a computação em nuvem e na borda, as fronteiras de segurança se turvam, aumentando a complexidade do uso de estratégias como segurança de perímetro por conta da heterogeneidade de tais ambientes. Além disso, empresas com cargas de trabalho sensíveis ainda têm que se preocupar com confidencialidade de dados em ambientes não confiáveis. O modelo de computação confidencial surgiu para atacar esse problema, garantindo confidencialidade em domínios não confiáveis através de execução criptografada de software.

Esses dois problemas parecem estar bem resolvidos quando considerados separadamente. No entanto, emitir identidades para cargas de trabalho confidenciais não é uma tarefa trivial, uma vez que ambientes de execução confiáveis têm processos de atestação muito particulares. Estratégias ditas *lift-and-shift* para computação confidencial têm seus serviços de atestação e configuração capazes de habilitar autenticação e comunicação segura entre suas cargas de trabalho confidenciais. Apesar disso, ainda há uma falta de um suporte de identidade que funcione para ambas cargas de trabalho confidenciais e não confidenciais, permitindo interoperabilidade entre esses dois tipos de cargas. Simplesmente utilizar *frameworks* de distribuição de identidades como o SPIFFE não é aplicável para computação confidencial, pois os modelos de ameaças de tais soluções assumem confiança na infraestrutura e na pilha de software onde as cargas de trabalho executam.

Neste trabalho, nós propomos uma integração entre um *framework* de distribuição de identidades nativo da nuvem com uma abordagem *lift-and-shift* que habilita computação confidencial. Foram integrados o *framework* SPIFFE e o SCONE para habilitar o suporte a identidades para cargas de trabalho confidencias que interromperam com cargas não confidenciais. Para isso, foi projetado um novo componente para o ambiente de execução do SPIFFE que entrega identidades para cargas de trabalho baseadas em SCONE e considera o modelo de ameaça de computação confidencial. Para avaliar nossa proposta sob uma perspectiva de segurança, foi conduzida uma análise de segurança com especialistas em computação confidencial para avaliar os pontos fortes e fraquezas da proposta diante do modelo de ameaça de computação confidencial. Os resultados da análise de segurança indicaram que os especialistas consideraram a proposta robusta contra ataques vindos da infraestrutura de um provedor em um ambiente não confiável. Além desses ataques, muitas ameaças de cargas de trabalho contra outras cargas de trabalho também foram consideradas mitigadas. Também foram conduzidos experimentos para identificar sobrecargas no processo de emissão de identidades. Por outro lado, experimentos para medir o tempo de construção de imagens de contêineres para cargas de trabalho escritas com Python mostraram que a construção de imagens SCONE é mais rápida. Apesar do escopo deste trabalho estar atrelado à cargas de trabalho baseadas em SCONE, a integração proposta é extensível para outras abordagens *lift-and-shift* através de plugins para o novo componente SPIRE.

Palavras-chave: Provisionamento de identidades; Computação confidencial; Intel SGX.

Abstract

Software no longer runs on an isolated set of server machines protected using perimeter security strategies and tools. As companies move to the cloud and edge computing, the security boundaries blur, turning strategies such as perimeter security and firewall management difficult in such heterogeneous environments. Also, companies with highly sensitive workloads still worry about data confidentiality in untrusted environments. The confidential computing model emerged to address this last problem, ensuring data confidentiality in untrusted domains via always encrypted execution.

These two problems seem to be solved when considered separately. However, issuing identities for confidential workloads is not trivial as trusted execution environments have strongly opinionated attestation processes. Confidential computing lift-and-shift approaches have their attestation and configuration services capable of enabling authentication and secure communication between confidential workloads. However, there is a lack of universal identity support between confidential and regular workloads. Simply using identity distribution frameworks such as SPIFFE to bootstrap identities for confidential workloads is not applicable because their threat model assumes trust in the infrastructure and software stack where the workloads run on.

In this work, we propose an integration between a cloud native identity framework and a lift-and-shift approach to enable confidential computing. We brought together SPIFFE and SCONE to enable identity support for confidential workloads that interoperates with non-confidential workloads. We designed a new component for the SPIFFE runtime environment that delivers identities for SCONE-based confidential workloads and considers the confidential computing threat model. To evaluate our proposal from a security perspective, we conducted a security analysis with confidential computing specialists to assess strengths and weaknesses under the confidential computing threat model. The security analysis results indicated that the specialists considered the proposal robust against coming from provider infrastructure in untrusted environments. Also, most workload-to-workloads attacks were considered mitigated. We also conducted experiments and identified overheads in the identity issuing process. On the other hand, experiments to measure container image build times for Python workloads showed that the builds with SCONE were faster than builds for non-confidential workloads. Although the scope of this work is tied to confidential SCONE-based workloads, the integration is extensible via plugins for the new SPIRE component and can accommodate other lift-and-shift solutions for confidential computing.

Keywords: Identity provisioning; Confidential computing; Intel SGX.

Acknowledgments

I am grateful.

I thank my parents, Paulo and Socorro, and my wife, Jéssica, for their support and encouragement.

I am also grateful to the Distributed Systems Lab for the pleasant environment and infrastructure provided. I thank the friends who helped me on this journey. Amândio, Ariel, Benardi, Eduardo, Fábio, João, Larissa, Leonardo, Marcus, Pedro, Ricardo, Rodolfo, Viviane, Whasley, thank you.

I thank the SPIFFE/SPIRE community for the support and discussions about the proposal.

I appreciate and acknowledge the immense contribution of experts during the security analysis. Andrey, Benardi, Clenimar, Eduardo, and Lucas, thank you for your contribution, patience, and perseverance in this process.

I thank my friends from the ELT information security team for the moments of learning and fun that helped me in many moments.

Last but not least, I thank my advisor, Andrey, for his advice and confidence in my work.

Thank you all.

Contents

List of Symbols

TEE - *Trusted Execution Environment*

SPIFFE - *Secure Production Identity Framework for Everyone*

SPIRE - *SPIFFE Runtime Environment*

TCB - *Trusted Computing Base*

NIST - *National Institute of Standards and Technology*

VM - *Virtual Machine*

SGX - *Intel Software Guard Extensions*

SDK - *Software Development Kit*

PRM - *Processor Reserved Memory*

TLS - *Transport Layer Security*

IAS - *Intel Attestation Service*

DCAP - *Data Center Attestation Primitives*

ECDSA - *Elliptic Curve Digital Signature Algorithm*

CVE - *Common Vulnerabilities and Exposures*

CAS - *Configuration and Attestation Service*

SVID - *SPIFFE Verifiable Identity Document*

URI - *Uniform Resource Identifier*

JWT - *JSON Web Token*

API - *Application Programming Interface*

CA - *Certificate Authority*

PKI - *Public Key Infrastructure*

SQUAD - *Secure Simple Storage Service for SGX-based Microservices*

RA-TLS - *Remote Attestation Transport Layer Security*

EPID - *Enhanced Privacy ID*

RCE	- *Remote Code Execution*
DoS	- *Denial of Service*
CSR	- *Certificate Signing Request*

List of Figures

List of Tables

Source Code List

Chapter 1

Introduction

Modern software no longer runs on an isolated set of server machines protected using perimeter security strategies and tools. As companies move to the cloud and edge computing, the security boundaries blur, making firewall management or perimeter security difficult in such heterogeneous environments. The Secure Production Identity Framework for Everyone (SPIFFE) is a framework that specifies standards to enable bootstrapping and issuing identities to services in such environments. Despite frameworks such as SPIFFE, companies with highly sensitive workloads still worry about data confidentiality in public cloud computing infrastructure. The confidential computing model emerged to address this last problem, using hardware-based trusted execution environment (TEE) technologies such as Intel Software Guard eXtensions (SGX) to ensure data confidentiality in untrusted domains via always encrypted execution, enabling confidential workloads.

1.1 Problem

There is a lack of universal identity support between confidential and regular workloads. In other words, there is a lack of identity systems that can issue identities for both confidential and regular workloads so that both types of workloads can communicate securely and establish a certain level of trust. Between confidential workloads using different trusted execution environment portability approaches the problem persists. There are lift-and-shift approaches to execute applications inside TEEs. However, each lift-and-shift approach, such as SCONE [Arnautov et al. 2016], has its attestation and configuration services capable of enabling confidential workloads to authenticate and communicate securely with other confidential workloads. In this sense, emitting identities for confidential workloads is not trivial as trusted execution environments have strongly opinionated attestation processes.

SPIFFE facilitates assigning identities in heterogeneous environments. Nonetheless, simply using SPIFFE [SPIFFE: Secure Production Identity Framework for Everyone. https://spiffe.io] to bootstrap identities for confidential workloads is not applicable because the SPIFFE threat model assumes trust in the infrastructure and software stack where the workloads run on. Thus, the attestation processes provided by the plugins available in the SPIRE implementation do not fit the trust requirements demanded by confidential workloads.

The incompatibility between the threats models of SPIFFE and confidential computing occurs mainly because, while SPIFFE assumes trust in components such as cloud providers and orchestrators for attestation, in confidential computing even the operating system is considered untrusted. While SPIFFE can rely on information coming from a container orchestrator, in the confidential computing world the only way is the attestation process defined by a TEE that has a much smaller attack surface.

Previous work addressed identity provisioning for confidential workloads. Proposed by Gregor et al. [Gregor et al. 2020], PALÆMON can attest and give identities for a subset of confidential workload. Due to the specific attestation process only suitable for confidential workloads, PALÆMON can not handle attestation for regular workloads. In previous work we proposed SQUAD [da Silva, de Oliveira Silva e Brito 2019], also capable of attesting a subset of confidential workloads and is not suitable for regular workloads. Knauth et al. [Knauth et al. 2018] proposed integrating a standard TLS communication channel with the Intel SGX remote attestation with the RA-TLS. With this integration workloads can share a common identity document (a certificate) but regular workloads should be modified to be aware of TEE attestation and RA-TLS does not provide attestation for non-confidential workloads. In general, the previous work present high adoption barriers and do not offer attestation for confidential and non-confidential workloads.

1.2 Intervention

In this work, we propose an integration between SPIFFE and SCONE to enable universal identity support for confidential workloads. As a well tested, ready-to-marked, lift-and-shift approach to run applications inside enclaves, SCONE lowers the barrier to adopting confidential computing and facilitates prototyping and experiments.

We designed a new component for the SPIFFE runtime environment (SPIRE) that enables SCONE-based confidential workloads and is robust against the threats defined in the confidential computing threat model. We evaluated our proposal in terms of security threats and performance. We conducted a security analysis with specialists in confidential computing to assess how which threats are mitigated by our proposal, according to the expertise of the analysis participants. The security analysis revealed that our proposal have an important impact considering the threat model for confidential computing. Most threats stated as coming from the infrastructure provider were considered mitigated by the participants. We conducted performance experiments that indicated overhead in the identity issuing process. Because of the stronger attestation process and encryption processes involved to protect workloads, issuing identities for confidential workloads takes longer than for regular workloads. Lastly, we also performed experiments to measure the overheads in the container image build times. In opposite to the on the identity issuing process, the experiments on container image build times showed that images for regular workloads written in Python take a longer build time than confidential SCONE images with applications written in the same language.

Although in this work we use SCONE-based workloads, the integration is extensible via plugins for SGX Helper, with the potential to accommodate other lift-and-shift solutions for confidential computing. By extending SPIFFE to confidential workloads, our proposal helps to push confidential computing towards cloud-native confidential computing.

1.3 Document structure

This work is organized as follows. In Chapter 2 we discuss base concepts and technologies needed to understand the proposal and its context. In Chapter 3 we present the motivations for this work. In Chapter 4 we present our proposal. In Chapter 5 we present a security analysis of our proposal, its results, conclusions, and lessons learned from it. In Chapter 6, we present the performance evaluation. In Chapter 7 we discuss the related work. Finally, in Chapter 8, we present the conclusions of this work, recent contributions and future work.

Chapter 2

Background

In this chapter, we provide background information on concepts and technologies to understand our work's context and contribution. Readers experienced in these topics may want to skip this chapter. Before going to more complex concepts, we are going to visit the concept of workload. In this work, we start from the definition of workload published by Feldman et al. [Feldman et al. 2020]: "A workload is a single piece of software, deployed with a particular configuration for a single purpose that can consist of multiple running instances, where all of them perform the same task." In the book, Feldman et al. emphasized that the term workload may include a wide range of different definitions of a software system. For our security analysis and experiments, we considered workloads limited to the boundaries of a container.

2.1 Trusted Execution Environments and Trusted Computing Base

A Trusted Execution Environment is an isolated processing environment that provides a specific set of guarantees. It provides isolated execution of applications (commonly called trusted applications), the integrity of the application code executing on the environment, the confidentiality of the data processed, and integrity of the trusted computing base (TCB) [Sabt, Achemlal e Bouabdallah 2015]. According to the National Institute of Standards and Technology (NIST) definition, a TCB is the totality of protection mechanisms within a computer system, including hardware, firmware, and software, the combination responsible for enforcing a security policy[1]. Moreover, a TEE may support remote attestation. Through the remote attestation process, the TEE can prove its trustworthiness and security properties to other entities.

Demigha and Larguet [Demigha e Larguet 2021] present a set of properties to consider for TEEs. They present several criteria in the perspective of cloud computing, from security criteria to functional and deployability criteria. For confidential workloads, security properties such as isolation, confidentiality protection, integrity protection, are must-have features. Moreover, in the context of cloud computing in untrusted environments, functional criteria such as TCB size and remote attestation support should also be taken into account.

The deployability criteria resume essential features which make it easier to deploy confidential workloads into a cloud-native environment. The use of legacy application code, the performance impact in TEE usage, virtual machine (VM) migration, and ecosystem size are essential criteria to look at.

2.2 Intel SGX

Intel Software Guard Extensions is a set of instructions and memory access control changes added to the Intel x86 architecture. Intel SGX is a hardware-assisted TEE that allows the creation of encrypted and isolated memory regions in the memory address space of an application. These protected memory regions, with access control enforced by hardware, are named enclaves. An SGX-enabled processor checks the operating system memory mapping, ensuring that only the right enclave instructions can access protected memory.

Unfortunately, the memory area dedicated to enclaves is still scarce compared to the main memory available in general-purpose computers nowadays. The memory region's typical size dedicated to enclave operations, Processor Reserved Memory (PRM), is 128 MB, and only a few recent processors have a maximum of 256 MB of PRM, but it should take a couple of years before they are standard. Once the processor needs to keep some metadata, the actual portion of memory useful to allocate enclaves is about 93 MiB, in systems with 128 MB of PRM. In VMs, these limits could be different. For instance, in Microsoft

[1]Source: https://csrc.nist.gov/glossary/term/trusted_computing_base. Last access in 23/03/2021

Azure, the flavors offer from 28 MB to 168 MB of Enclave Page Cache (EPC) [Microsoft Azure: Confidential Computing Documentation.]. If the data inside an enclave exceed the available memory amount, the processor evicts the protected memory pages into the main memory. This process increases the memory access latency in orders of magnitude, so developers should design enclaves with these limits in mind to avoid the risk of considerable overhead [Arnautov et al. 2016]. Although the current scarce protected memory amount, the new third-generation of the Xeon scalable processor (Ice Lake SP) came with SGX and total memory encryption capabilities. It will enable running workloads without the secure memory swap overhead.[2]

SGX applications are composed of one untrusted part and at least one enclave (trusted part), that is, an SGX application has one untrusted portion but can have several independent enclaves. As an enclave can not perform system calls due to the TEE isolation properties, the untrusted part of the application is always responsible for input and output operations and instantiating enclaves. In any case, enclaves must be designed never to leak sensitive data in plain text.

The most distinguishing features of Intel SGX are memory protection and remote attestation. Memory protection guarantees both confidentiality and integrity. Once the application code is loaded into the so-called SGX enclave, the code cannot be modified or inspected, even by code with higher privileges, such as the operating system. Furthermore, if this code receives data through protected channels, such as Transport Layer Security (TLS) connections, the data cannot be stolen. The implementation is based on new instructions and units added to the processor itself, which enable higher security levels than what can be provided by software alone [Costan e Devadas 2016].

The confidentiality and integrity protection of the enclaves are then completed by the second major feature: remote attestation. The attestation process is a mechanism by which a challenger can gain confidence that an enclave is running in an SGX-enabled platform [Scarlata et al. 2018]. Also, through this process, a challenger can verify the identity associated with an enclave. The enclave identity, also known as MRENCLAVE, is a SHA-256 digest of an internal record of all the activities done while the enclave is built [Anati et al. 2013].

[2]Ice Lake SP technical documentation: https://www.intel.com/content/www/us/en/products/platforms/details/ice-lake-sp.html.

Thus, the digest includes all pages of code, data, stack, the heap, relative position of the pages, and all security flags associated. Intel currently provides two approaches to perform remote attestation. The first one, using the Intel SGX Attestation Service (IAS), is a client platform focused approach that uses a privacy-preserving group signature scheme that is only verifiable by IAS. The second one, the Intel SGX Data Center Attestation Primitives (Intel SGX DCAP), is based on Elliptic Curve Digital Signature Algorithm (ECDSA) signatures [Scarlata et al. 2018]. These primitives allow the construction of on-premise attestation services, enabling third parties to build their attestation infrastructure, especially useful for enterprises that do not want to outsource trust decisions to Intel and for environments where internet access latencies need to be avoided.

Figure 1 depicts the default remote attestation flow. In step 1, the challenger application challenges the SGX application to prove that it is running in an SGX-enabled machine. The challenge also has a nonce, for freshness purposes. As the trusted part of the SGX application cannot perform system calls, all the communication between the challenger and the enclaves pass by the untrusted part, but end-to-end encryption ensures confidentiality and message authentication ensures integrity of messages exchanged during the execution of the attestation process. In step 2, the untrusted part requests an attestation report from the enclave, passing the nonce sent by the challenger. The enclave generates the attestation report and a manifest, and sends it back to the untrusted part (step 3). The untrusted part delivers the generated report to the Quoting Enclave for signing (steps 4 and 5). The Quoting Enclave is a special enclave provided by Intel and has access to signing keys that never leave the CPU. The report signed with this secret key is forwarded back to the challenger (step 6), which can validate it with an attestation service, IAS or DCAP, as mentioned above (step 7). Once the quote can be trusted (step 8), the digest identifying the application can be compared against known reference values (step 9).

2.3 SCONE

SCONE is a runtime environment and set of tools to run workloads inside SGX enclaves. To achieve this, it provides a lift-and-shift approach. This approach makes SCONE a better option than the SGX SDK to port existing workloads to SGX enclaves. To run workloads in-

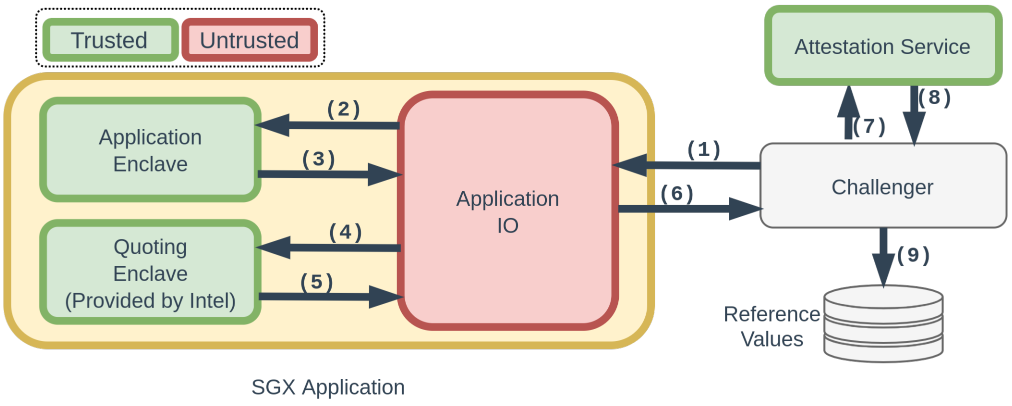


Figure 2.1: Intel SGX Remote Attestation Flow

side enclaves using the SGX SDK, a developer must rewrite the code using specific libraries with limited features. Also, the software development kit limits the developer to C and C++ programming languages.

On the other hand, SCONE leverages modified versions of the standard libraries such as `libc` and `musl-libc`, supporting various programming languages (C, C#, C++, Go, Python, Java, Rust, among others). It offers secure containers on top of an untrusted software stack, protecting the workloads even against high privileged agents that control any part of this untrusted software stack [Arnautov et al. 2016]. The SCONE environment provides an external interface for systems calls protected by shields. These shields ensure the confidentiality and integrity of data passing through file descriptors. Also, SCONE integrates with Docker and container management platforms compatible with Docker, such as Kubernetes, Docker Swarm, and Docker Compose.

To develop with the SGX SDK, the developer has to consider security advisories and common vulnerabilities and exposures (CVE). In the case of using SCONE, the security issues related to SGX are concerns of the SCONE environment, which allows the developers to focus on the functionality of their workloads. For instance, the SCONE has available protection against practical side-channel attacks, with one approach using Varys [Oleksenko et al. 2018].

SCONE also supports remote attestation. The SCONE environment has a service that integrates the remote attestation process and secrets provisioning. The Configuration and Attestation Service (CAS) enables operators to manage secrets to their confidential workloads with rollback protection and updates in a secure way. Also, the CAS is attestable and allows operators to verify its integrity.

Gregor et al. [Gregor et al. 2020] explain that one of the challenges in the CAS design process was "how to support secret management for common configuration approaches in a secure way and without requiring modifications to the source code?" Not changing the application's code was a requirement since SCONE proposes a lift-and-shift approach. The solution proposed by the authors covers then three popular ways of configuration provisioning: command-line arguments, environment variables, and files.

Using CAS, an operator can describe the workload, properties, security constraints, and the configuration for this workload. Then, the operator deploys the workload remotely and, to receive the configuration, the workload goes through an attestation process that ensures the properties and security constraints. In this way, the CAS verifies the enclave identity and the properties of the SGX TCB.

All information that describes a workload is loaded into CAS using security policies commonly referred to as sessions. In this work, we will use sessions to refer to security policies. A session is written in a policy language that is a subset of the YAML languange. The code snippet 2.1 shows an example of a session for a Python workload. The example illustrates the secrets provisioning in three different ways: via command line arguments (line 7), via environment variables (lines 8 and 9), and via files injected in the workload's view of the filesystem (line 22).

Source Code 2.1: Session example for a Python workload.

```
 1 name: python-workload-session
 2 version: "0.3"
 3 services:
 4  - name: python-workload
 5    image_name: my-image
 6    mrenclaves: [708
         bfa35c15da4767700cad3dc49e7002c4914f452f487973d589dc4e000ed03
         ]
 7    command: /usr/bin/python3 /app/workload.py arg1 arg2
 8    environment:
 9       MY_ENV_VAR: my-env-value
10
11 secrets:
12  - name: my-private-key
13    kind: private-key
14    value: |
15       -----BEGIN PRIVATE KEY-----
```

```
      ...
      -----END PRIVATE KEY-----

images:
 - name: my-image
   injection_files:
    - path: /keys/client-key.key
      content: $$SCONE::my-private-key.key$$

security:
   attestation:
      tolerate: [debug-mode, hyperthreading, outdated-tcb, insecure
         -igpu]
      ignore_advisories: "*"
```

The documentation for the SCONE policy language is available on the SCONE website [3]. In our example, we describe the session `python-workload-session` as follows:

- The key `name` holds the session's name.
- The key `version` indicates the version of the policy language for the current session.
- The section `services` describes the workloads, with the following subkeys:
  - `name` indicating the name of the workload;
  - `image_name` to specify which SCONE image (view of a filesystem provided by the SCONE environment) the workload will see;
  - `mrenclaves` listing the enclave identities allowed to receive the secrets described in the current session;
  - `command` specifying which workload will run and with which command-line arguments;
  - And `environment` defining which environment variables the workload process will have access.
- The section `secrets` defines secrets in general (in this case, a PEM-encoded private key).

[3]https://sconedocs.github.io/CAS_session_lang_0_3/

- The `images` section describes views of a filesystem available to the workload after the attestation process. In this case, the workload will be able to read the file `/keys/client-key.key` with the content stored in secret named `my-private-key`.
- The section `security` describes constraints about the platform: which features of the TCB the CAS should tolerate (key `tolerate`) and which security advisories should be ignored (key `security advisories`). Typically this section is used to weaken some constraints for development and test environments.

The CAS has strong access control policies and a scheme in which sessions can import secrets of other sessions, enabling various scenarios for operators such as reuse and secret compartmentalization. In this work, we use this feature to give our proposal components only the necessary views to work correctly.

Another import SCONE feature is the File System Protection File (FSPF). The FSPF allows the encryption with authentication of files within the Docker images built for the applications. With this feature interpreted code and dependencies can be encrypted to ensure code confidentiality. Moreover, the FSPF has a authenticated-only mode useful for cases where only integrity is a requirement. This feature can be used to protect files created by the applications in runtime ensuring that no information will leave enclaves without the proper security measures.

2.4 Secure Production Identity Framework for Everyone

The following sections present the SPIFFE standard and its reference implementation, SPIRE.

2.4.1 SPIFFE

The Secure Production Identity Framework For Everyone (also referred as SPIFFE by the community) is a framework and a set of standards for identifying and securing communications between application services [SPIFFE: Secure Production Identity Framework for Everyone. https://spiffe.io]. SPIFFE defines how services identify themselves to each other, establishing a standard. It allows workloads to get identities and to trust each other without the need for a pre-existing secret.

SPIFFE consists of several parts: The SPIFFE ID, the Trust Domain, the Federation, the Trust Bundle, the SPIFFE Verifiable Identity Document (SVID), and the Workload API.

The SPIFFE ID is the string representation of the identities issued to services or workloads. It follows the Uniform Resource Identifier (URI)[4] scheme and consists of several parts: the prefix (as the URI's scheme), the name of the trust domain (as the host component), and the workload's identity (as the path component). An example of SPIFFE ID could be `spiffe://example.com/myworkload`.

Operators use the SPIFFE Trust Domains to manage administrative and security boundaries within the organizations. The Trust Domain works like a namespace. Each Trust Domain has its issuing authority, which means that one Trust Domain's compromise does not necessarily compromise another.

The SPIFFE Federation gives to SPIFFE implementations the ability to validate SPIFFE IDs across Trust Domains. Use cases like different organizations that need to communicate with each other or even a single organization that needs various security boundaries can benefit from this SPIFFE feature. To make this happen, operators can configure foreign Trust Domains' Trust Bundles to be available through TLS-protected endpoints.

The SPIFFE Trust Bundle is a document that contains the Trust Domain's public keys. Thus, every Trust Domain has a Trust Bundle associated.

The SVID is the cryptographically-verifiable document that wraps the SPIFFE ID. Each SVID has a single SPIFFE ID. The Trust Domain's issuing authority signs each SVID, so it is verifiable using the Trust Bundle. Moreover, at the time of this work, there are two encoded forms for SVIDs. The X509-SVID wraps a SPIFFE ID into a standard X.509 certificate [5]. The JWT-SVID encodes a SPIFFE ID into a standard JSON Web Token (JWT) [6].

Lastly, the SPIFFE Workload API is a local Application Programming Interface (API) that workloads use to get their SVIDs and Trust Bundles. The Workload API is a particular API that uses a UNIX socket (non-networked) to access a gRPC server. The Workload API is unauthenticated, which means that a workload does not have to know any pre-existing

[4]URI RFC: https://tools.ietf.org/html/rfc3986. Last access in 23/03/2021

[5]X.509 RFC is available at https://tools.ietf.org/html/rfc5280

[6]JWT RFC is available at https://tools.ietf.org/html/rfc7519

secret. Considering the pre-existing threat model for the SPIFFE implementations, this API allows the Agent's workload attestation plugins to identify callers without the need for direct authentication. Unlike the SGX attestation process, the SPIFFE attestation process identifies workloads through a set of features collected by attestation plugins using out-of-band mechanisms. If the features match with the features in one or more entries registered, the caller receives the identities for that set of entries. For instance, a possible set of workload features could include:

- The user ID;
- The Kubernetes Pod label;
- The Kubernetes Service Account;
- The Kubernetes namespace where the workload runs.

The caller processes are identified by their process ID and the attestation plugins implement the methods to get the processes features. The Kubernetes workload attestor queries the Kubernetes API to get this information. The UNIX workload attestor looks at the `/proc` directory to get process features.

2.4.2 SPIRE

The SPIFFE Runtime Environment (also referred as SPIRE by the community) is the reference implementation of SPIFFE. It's an open-source project hosted by the Cloud Native Computing Foundation. SPIRE consists of two main parts named SPIRE Server and SPIRE Agent. Figure 2.2 depicts SPIRE architecture.

The SPIRE Server manages a SPIFFE Trust Domain, issuing all identities. To do this, it holds information about its SPIRE Agents and workloads. The Server attests Agents via a node attestation API using an extensible set of attestation plugins. In addition to attestation, the SPIRE Server has other plugins responsible for different tasks. There are also plugins for configuring upstream signing authorities and for data storage.

On the other hand, the SPIRE Agent has a single concern. It is responsible for exposing the Workload API to workloads. The SPIRE Agent also is extensible by using plugins. There

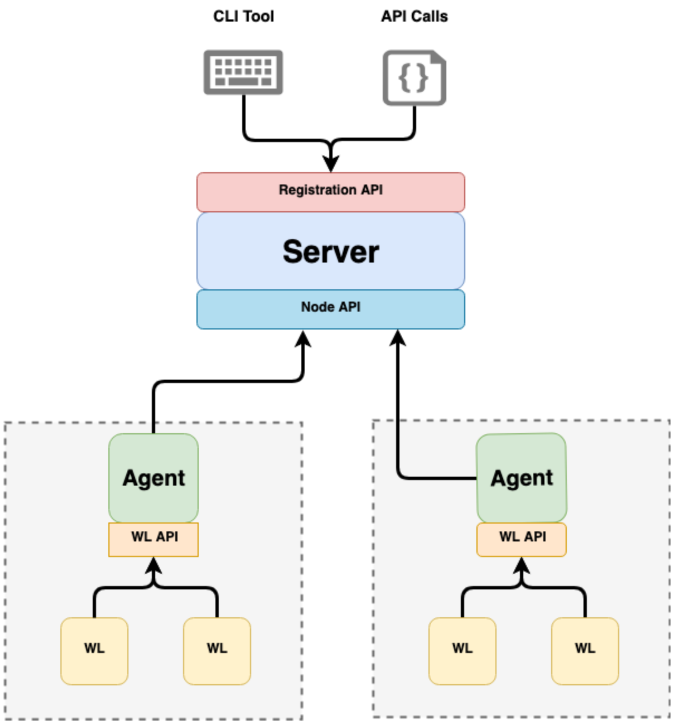


Figure 2.2: SPIRE architecture. Source: `https://spiffe.io/docs/latest/spire-about/spire-concepts/`. Last access in 22/03/2021.

are plugins for its attestation process with the SPIRE Server and plugins to attest several kinds of workloads via the Workload API.

Figures 2.3 and 2.4 illustrate the pluggable feature of the SPIRE implementation, expanding the architecture view.

The SPIRE server supports a variety of plugins that define its functionalities. The Server can upstream authority plugins to inherit a certificate authority (CA) from another Public Key Infrastructure (PKI) system. Despite this plugin type, the Server is capable of work as a CA. The Server uses key manager plugins to store private keys used to sign its SVIDs. The data store plugins store all other data that the SPIRE Server has to keep to work properly.

The Server uses the node attestor plugins to ensure properties and verify nodes running SPIRE Agents. As a result of the node attestation process, the Server gets selectors (features built using the attestation plugins) for the node running the Agent.

On the other hand, the SPIRE Agent has fewer plugin types. It also uses a key manager plugin and must use node attestation plugins that work together with the Server's node attestation plugins to identify the Agent.

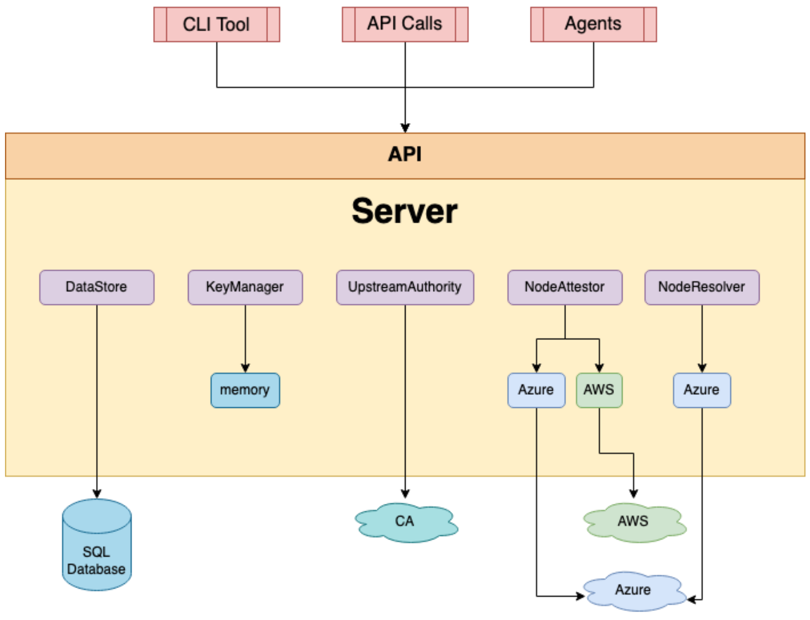


Figure 2.3: View of SPIRE Server pluggable interfaces. Source: `https://spiffe.io/docs/latest/spire-about/spire-concepts/`. Last access in 22/03/2021.

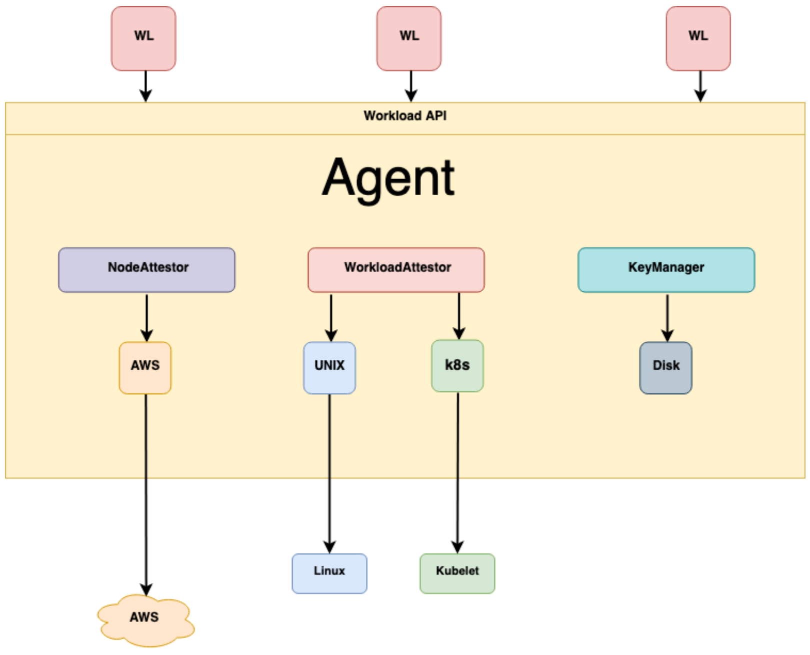


Figure 2.4: View of SPIRE Agent plugglabe interfaces. Source: `https://spiffe.io/docs/latest/spire-about/spire-concepts/`. Last access in 22/03/2021.

However, the most important plugin type for the SPIRE Agent is the workload attestor. In the workload attestation process, when a workload demands an identity, the workload attestation plugins communicate with resources available in the node, e.g., the operating system kernel or the Docker engine, to determine properties of the calling workload to decide if it will provide an identity and what identity is this.

2.5 Cloud Native and Cloud Native Workloads

Cloud-native is an approach for the development and deployment of workloads that leverages the cloud computing model. Gannon et al. [Gannon, Barga e Sundaresan 2017] enforce some properties of cloud-native applications that help to describe cloud-native applications.

Cloud-native applications can operate globally and can have data and services replicated in different geographic regions. Cloud-native applications should scale well with many concurrent users and are designed assuming that the infrastructure is prone to failure. Moreover, developers plan these applications so that the rolling out process and testing occur without breaking production environments, even if the new version crashes.

Nowadays, the microservices paradigm is one of the most used to design cloud-native applications. It divides the application into small, independent components, referred to as microservices. It facilitates various processes from the deployment to the daily operation of the application. Ideally, each microservice has a single concern, working together with others to solve a bigger problem.

In general, microservices are packaged in containers. Some tools facilitate the scaling and management of these workloads and can watch the microservices, restart them when they fail, and increase or decrease the number of replicas according to the incoming load. Also known as orchestrators, these application deployment tools are available as fully managed services in most public cloud providers such as Google Cloud, Amazon Web Services, Microsoft Azure, Digital Ocean, etc. All the public cloud provides cited offer a Kubernetes service. Finally, cloud customers can themselves deploy and manage a Kubernetes service to orchestrate their workloads. Having microservices increases the number of independent components in a distributed system. Thus, scaling and migration make common strategies difficult. The use of an identity management system such as the SPIFFE framework helps in this problem.

Confidential cloud-native workloads present almost the same features as cloud-native workloads. In general, the downside of using TEEs to protect workloads and data is the performance loss. However, by having TEE-enabled microservices, one can have encrypted execution and horizontal scaling, minimizing the performance loss's impact.

Chapter 3

Motivations to integrate SCONE and SPIRE

This chapter details the threat model for confidential workloads, the threat model for SPIFFE and SPIRE, and the motivations to integrate confidential workloads and the SPIFFE reference implementation.

3.1 Threat Model for Confidential Computing

The threat model for confidential workloads (either developed with Intel SGX SDK or using a lift-and-shift approach) assumes a powerful attacker with superuser privileges on the infrastructure where the applications run. Therefore, an attacker can control the entire software stack, including operating systems, hypervisors, and cloud computing platform software. An attacker can use these powers to steal cryptographic keys or data, or even change application code or binaries.

The motivations for such an aggressive threat model are various. First, an attacker can steal operator identities during attacks such as spear phishing. Besides that, operators or providers could be obligated by law to provide data. Even if the provider and operators were trusted, the software stack has a large and complex codebase. Second, underestimating the threat model could be expensive. The average cost of a data breach is about US$ 3,86 million, with detection and contention of 280 days, on average [IBM Security. Cost of a Data Breach Report 2020. https://www.ibm.com/security/data-breach].

Thus, assuming that the infrastructure and the software stack is untrusted, the threat model reduces the TCB to the SGX-enabled applications, the SGX hardware, and software running on trusted-by-assumption instances (e.g., the admin's own machine). The threat model also assumes that trusted applications are free of bugs and side-channel attacks are out of scope. This assumption has some impacts that depend on the way that developers design trusted applications. For code developed with the Intel SGX SDK, the developers have total control over the application partitioning (the trusted parts and the untrusted part) and must implement mechanisms to reduce risks. For code developed with a runtime, SCONE in this proposal, mechanisms implemented by the runtime mitigate risks in the applications. An example is Varys [Oleksenko et al. 2018] that mitigates side-channel attacks by detecting abnormal interruptions in the enclave execution and other strategies.

3.2 Threat Model for SPIFFE and SPIRE

The SPIFFE/SPIRE threat model assumes that network communications are hostile and potentially fully compromised. Furthermore, in opposition to the SGX, this threat model assumes that the hardware on which SPIRE components run, as well its operators, are trustworthy. In addition to the trust in hardware, some methods of node and workload attestation may imply in third-party platforms or software. It means that leveraging AWS or Google Cloud based node attestation plugins implies trusting the respective computing platforms. Similarly, leveraging Kubernetes for workload attestation implies that the Kubernetes is assumed to be trustworthy.

Feldman et al. [Feldman et al. 2020] defined the threat model by determining the security boundaries between SPIRE core components: workload, Agent, and Server. In addition to the network communications, workloads and Servers in other trust domains are untrusted.

The trust boundary between the Agents and workloads implies that Agents do not trust any information sent directly from workloads. In this sense, all the attestation mechanisms consider only out-of-band ways to attest workloads. This implication affects one of the solutions presented later in chapter 4.

Another trust boundary is defined between Agents and Servers. While Agents are more trusted than workloads, Agents are less trusted than Severs. There are mechanisms in place to limit the impact of a compromised node. The Parent ID is an identity associated with all workload identities. It allows specifying which identities are under the responsibility of a certain Agent, imitating the set of identities an Agent is allowed to emit and keep in cache.

The last trust boundary is placed between Server of different trust domains. SPIFFE allows a Server to mind SVIDs only within its trust domain. If a Server federates with others it can not mint identities on behalf of other trust domains. This boundary exists only between different trust domains because SPIRE has not multi-party protection. All Servers in the same trust domain have access to the same signing keys.

3.3 The Integration

There is a lack of a robust attestation process in SPIFFE capable of ensuring integrity and trust for applications in an untrusted environment. Even in zero-trust solutions like SPIFFE, an attacker with superuser privileges, as described in the confidential computing threat model, can mint attestation evidence. Thus, the attestation processes provided by the plugins available in the SPIRE implementation do not fit the trust requirements demanded by confidential workloads.

However, we can leverage SPIFFE/SPIRE to give confidential workloads universal identity support based on the SPIFFE standard. With SPIFFE identities, confidential workloads can communicate with other confidential workloads or even non-confidential workloads securely. Workloads then can give each communicant different levels of trust based on which SPIFFE ID is presented. It allows the abstraction of the attestation process between workloads, trusting only in the identities received in these communications.

Nevertheless, to integrate with confidential computing, the SPIRE implementation should be aware of the more aggressive threat model that comes with confidential workloads. It may impose extra security shields or new specialized components in the SPIRE implementation. In this sense, we need a security evaluation to ensure that using SPIRE common attestation plugins does not fit the threat model for confidential computing. Besides, a security evaluation can point the pros and cons of a proposal to give universal identity support for confidential workloads using SPIFFE/SPIRE.

In this work, we propose a novel approach to integrate SPIRE and confidential workloads.

We also present a security analysis to enforce the need for this new approach. Finally, we show the results of experiments on the time needed to get a new identity using our solution, demonstrating that overheads are not prohibitive.

Chapter 4

Confidential Workloads and SPIFFE: Integration Proposal

Considering the motivations discussed in Chapter 3, we designed two approaches to give support for confidential workloads to SPIFFE, leveraging its reference implementation (SPIRE).

The first approach is based on a new workload attestation plugin. Once a workload starts up, it should get attested to retrieve its initial configuration and a piece of secret information (challenge), unique for that workload. Then, it can call the Workload API and pass the challenge to be verified by the SCONE workload attestation plugin. The attestation plugin checks the secret within CAS and returns the session name and the session hash as selectors to be used by the Agent core, which will decide what identities will be delivered to the workload.

The second and final approach, called SGX Helper, is an alternative to the SCONE workload attestation plugin because of first approach, regarding the specification of the Workload API. The Workload API specification mandates that workloads must not send any secret or pre-authentication artifact to the Workload API. In this sense, the challenge sent by the SCONE workload, in the first approach, is not compliant with the Workload API specification.

The SGX Helper is a new specialized SPIRE component capable of providing identities for confidential workloads by pushing them to trusted secret stores.

In Section 4.1, we detail the workload attestation plugin approach. In Section 4.2, we present the approach we have chosen to perform a security analysis and performance experiments. We illustrated the approaches workflows using the Figures 4.1 and 4.2. To facilitate the visualization, the Figures show the trust boundaries based on which location we placed these components. The admin site is a hardened location, trusted-by-assumption: a private cloud, data center, or on-premise machines. One would place the SPIRE server in the admin site because of the consequences of compromising this component. The untrusted portion is an environment (site or machine) controlled by third parties, typically a public cloud. Finally, in shades of green, the TEE represents the SCONE trust execution environment's trust boundaries.

4.1 SCONE Workload Attestation Plugin

This approach consists of using a workload attestation plugin and challenging the confidential workload to prove that a trusted CAS successfully attested it. Figure 4.1 depicts the identity issuing process workflow using the SCONE workload attestation plugin.

Figure 4.1 and Figure 4.2 depict the trust boundaries based on which location we placed these components. The admin site is a hardened location, trusted-by-assumption: a private cloud, data center, or on-premise machines. One would place the SPIRE server in the admin site because of the consequences of compromising this component. The untrusted portion is an environment controlled by third parties, typically a public cloud. Finally, in shades of green, the TEE represents the SCONE trust execution environment's trust boundaries.

The detailed workflow is as follows. In step 1, the operator posts the session with the initial configuration for the workload, including the constraints that must be satisfied in the SGX attestation process. In step 2, the operator also posts the challenge session. It is a session with a single secret shared imported by the session in the previous step. In other words, the workloads will only get the secret if it got the initial session through the attestation process. In step 3, the operator registers the workload using the first session's name and hash digest. The SCONE CLI's create session command returns an SHA-256 digest that identifies the session created. After the start-up completes, the SPIRE Agent will cache the registration entries under its responsibility in step 4. The operator then deploys the workload, and in steps 5, 6, and 7, the workload gets attested and receives the challenge along with other initial configurations. The workload then uses the challenge (and its session name) in a call

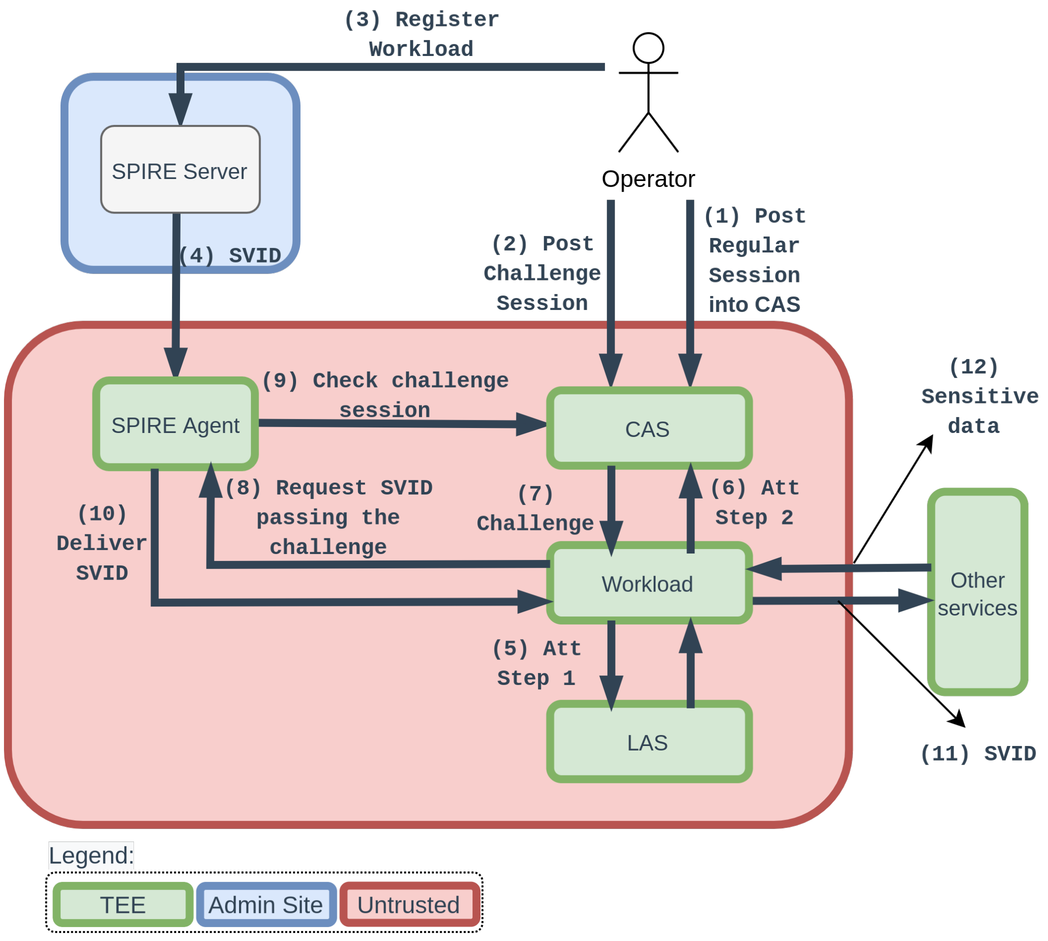


Figure 4.1: Identity issuing process workflow using the SCONE workload attestation plugin.

to the Workload API (step 8), asking for an SVID.

In step 9, the SPIRE Agent enabled with the SCONE workload attestation plugin checks the challenge session within CAS. If the challenge value passed by the workload matches the session stored in CAS, the plugin builds the right selectors, and the SPIRE Agent returns an SVID in step 10. With the SVID, the workload can communicate with other services using its SPIFFE ID. We illustrate this communication in steps 11 and 12.

Code sample 4.1 shows an example of a session that the operator posts into CAS. The file format is YAML, and lines 18 to 22 show how the initial session (posted in step 1) imports the secret from the challenge session. Line 13 shows that the SCONE environment will inject the secret into the workload process as an environment variable. Line 9 defines the expected MRENCLAVE for the workload.

Source Code 4.1: Session example

```
 1 # session.yaml
 2 name: session-sgx
 3 version: "0.3"
 4
 5 services:
 6  - name: myapp
 7    image_name: svid-image
 8    mrenclaves: [
 9    8144dc20883a9341bac687715aea6f50373238a4388a1670301d8f5b438190d0
10    ]
11    command: python3 /app/example.py
12    pwd: /app
13    environment:
14      SVID_CHALLENGE: $$SCONE::svid-challenge$$
15      RESOURCE_ADDR: "https://192.168.1.102:5555"
16      CAS_SESSION_NAME: "spire-svid-session-sgx"
17
18 secrets:
19  - name: svid-challenge
20    import:
21      session: spire-svid-session-sgx
22      secret: svid-challenge
23
24 security:
25   attestation:
26     tolerate: [hyperthreading]
27     ignore_advisories: []
```

Code sample 4.2 shows the challenge session (posted in step 2). This session describes a secret which the name is svid-challenge, and the value is `deadbeef`. Besides, lines 8 to 10 define the access rule to that secret. In this case, only the session named `session-sgx` can import this secret. Still, if the session's name matches, but its hash digest does not, the access is denied. Then, this hash prevents unauthorized or accidental updates in the importing session constraints. Also, with this two-sessions solution, the workload attestation plugin has no access to the secrets and configurations for the workloads in CAS.

Source Code 4.2: Session example: challenge session.

```
 1 # challenge.yaml
 2 name: spire-svid-session-sgx
 3 version: "0.3"
 4 secrets:
 5   - name: svid-challenge
 6     kind: ascii
 7     value: deadbeef
 8     export:
 9       session: session-sgx
10       session_hash: ${SESSION_SGX_HASH}
11 access_policy:
12   read:
13     - CREATOR
14     # Workload attestor certificate
15     - |
16       -----BEGIN CERTIFICATE-----
17       -----END CERTIFICATE-----
18   update:
19     - CREATOR
```

After the initial designing process of each approach, we submitted the design to the SPIFFE/SPIRE community validation. In the case of the workload attestation plugin, we identified an incompatibility with the Workload API specification[1]. It mandates the absence of direct client authentication. In this way, the Workload API implementation does not pass through any metadata from its callers to the workload attestation plugins. Moreover, this implementation is not prone to changes. Thus, the only way to proceed with this approach is to have a version of the SPIRE Agent non-compliant with the SPIFFE specification, which is infeasible. With this conclusion, we proceeded to the security analysis and experiments considering only the second approach, the SGX Helper.

[1]Workload API standard: https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE_Workload_API.md#41-identifying-the-caller

4.2 SGX Helper

The second approach that we considered for this work is implementing a new SPIRE component specialized in handling identities for confidential workloads. We named the new component SGX Helper. We designed the component and developed a proof-of-concept by forking the SPIRE Agent codebase for simplicity and faster development. Considering the scope of this work, we implemented only the integration with the SCONE environment. However, this component can be as extensible as the SPIRE Agent with the help of custom plugins for different solutions for running SGX workloads.

Considering the threat model for confidential computing and the remote attestation process supported by SCONE, an SGX Helper deployment in the same node as the confidential workload is not needed. In this sense, the SGX Helper can run in the trust zone we call the admin site. For minimal impact on both SPIRE and SCONE codebase and usage experience, the SGX Helper can fetch SVIDs under its responsibility, working similar to an Agent with a parent identity. With the registration entries in hand, the SGX Helper can push the identities to the SCONE CAS service, responsible for the attestation process and delivery of identities to the confidential workloads.

Even using the SGX Helper, the operator still needs to provide an initial configuration to the workload. The workload identity will come from the SPIRE system. However, as in the first approach, the SGX Helper must not read the workload initial configuration. Consequently, two sessions will be created.

The operator creates the first session, which defines the initial parameters. The SGX Helper generates the second session containing the SPIFFE Verifiable Identity Document. The first session has information about remote attestation constraints, such as enclave MRENCLAVEs and the SGX platform properties. Nevertheless, the session associated with the SPIFFE IDs needs also to consider the workload configuration, as different configurations could imply different identities or security levels. Therefore, to issue identities for SCONE-SGX confidential workloads, two special selectors are still available in the second approach:

- `cas_session_name`: name of the session posted into SCONE-CAS by the operator.

- `cas_session_hash`: hash of the session posted into SCONE-CAS by the application operator. This hash digest enforces not only the MRENCLAVEs that are acceptable but the integrity of any configuration that is passed through an environment variable, configuration files, or through the command line itself.

The SGX helper will push the SVIDs into the SGX secret store (CAS), restricting the access only to applications under the constraints and configurations summarized in the session with a name equals to the specified in the session name selector, and a hash digest equals the digest specified in the session hash selector (`cas_session_hash`). With these constraints, only attested workloads (matching the CAS sessions) will get SVIDs.

Figure 4.2 depicts the SVID issuing workflow. In step 1, the operator posts a session into CAS with the base configurations for the workload. In step 2, the operator registers the workload using the SPIRE Server registration API. The operator creates the registration entry using the two selectors described above that unambiguously describe the workload. In step 3, the SGX Helper fetches the SVIDs under its responsibility and then pushes the SVID and its updates into the secret store (CAS). At any moment after the registration, the operator can deploy the workload (step 5).

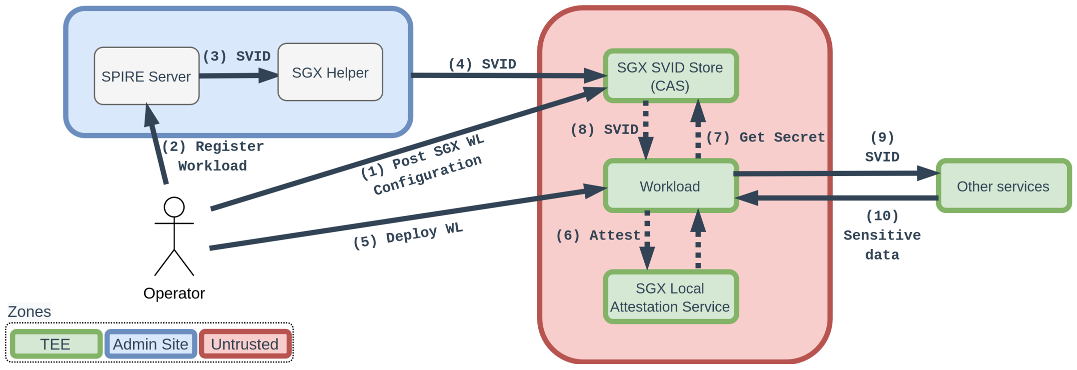


Figure 4.2: Identity issuing process workflow using the SGX helper.

In the workload startup, the enclave gets attested (steps 6, 7, and 8), transparently receiving both the initial configuration defined by the operator and the SVID cached in the secret store. According to how the application administrator configured the application (in step 1), the SCONE environment delivers these pieces of information to the application through environment variables, files visible only to the application, or even through strings injected as command line parameters.

Code snippet 4.3 shows a sample of a session posted by the operator in step 1. Code snippet 4.4 shows the message pushed by the SGX Helper into the SGX secret store (CAS).

Source Code 4.3: Session with the initial workload configuration posted by the operator.

```
name: session-sgx
version: "0.3"
services:
 - name: workload
   image_name: svid-image
   mrenclaves: [ ${WORKLOAD_MRENCLAVE} ]
   command: python3 /app/app.py
   pwd: /app
   environment:
     APOLLOS_MSG: "Hello Secure World!"

secrets:
 - name: svid
   import:
     session: spire-svid-session-sgx # same session name prefixed
        with spire-svid-
     secret: svid

images:
 - name: svid-image
   injection_files:
    - path: /certs/svid.crt
      content: $$SCONE::svid.crt$$
    - path: /certs/svid.key
      content: $$SCONE::svid.key$$
```

Source Code 4.4: Session with SVID as payload. Posted by the SGX Helper.

```
# Certificate and private key truncated to improve readability
name: spire-svid-session-sgx
version: "0.3"
secrets:
  - name: svid
    kind: x509
    value: |
        -----BEGIN CERTIFICATE-----
        -----END CERTIFICATE-----
    export:
        session: session-sgx
        session_hash: ${SESSION_SGX_SHA256}
    private_key: svid_key
  - name: svid_key
    kind: private-key
    export:
        session: session-sgx
        session_hash: ${SESSION_SGX_SHA256}
    value: |
        -----BEGIN PRIVATE KEY-----
        -----END PRIVATE KEY-----
```

For simplicity in explaining the new component, we omitted the push of the Trust Bundle and Federated Trust Bundles. The new component also pushes these CA certificates (when available) to the secret store (CAS). Moreover, the access control over these bundles is more flexible since the bundles' content is not sensitive as the content of SVIDs. Also, workloads can continuously pull bundles from the CAS secrets API. Also, the session templates are configurable.

Chapter 5

Security Analysis

The SPIFFE/SPIRE team worked on a security analysis to understand potential security risks to common SPIFFE deployment scenarios. The goal of this self-assessment was to strengthen SPIFFE/SPIRE security. The specialists organized the results in a document describing the project design, intended use, and the security analysis. They organized the security analysis results in a set of matrices, one for each attack type resultant of the analysis process, and some general recommendations.

A security analysis gives a notion of the threats to a deployment (within the security analysis limitations). Moreover, it enables us to rank the threats and prioritize the effort. In this work, we leveraged the SPIRE's security analysis to study the trade-offs of our proposal to integrate SCONE confidential workloads with SPIRE. The security analysis also presented reliable evidence that the current analyzed SPIRE implementation (our base case) does not fit the confidential workload's threat model requirements.

This chapter describes the initial security analysis performed by the SPIRE team and the security analysis conducted by us. Moreover, it presents the results of the security analysis and some considerations about these results.

In Section 5.1 we describe the security analysis performed by the SPIRE team. In Section 5.2 we present the base case to contrast with the proposed to integrate SPIRE and confidential workloads. Then, in Section 5.3 we describe the our security analysis. In Section 5.4 we present the security analysis output matrices and , in Section 5.5, we present conclusions about these outputs. Finally, In Section 5.6 we present our considerations about the challenges faced in the security analysis process.

5.1 Previous SPIRE Security Analysis

This section describes the initial SPIRE security analysis based on the output artifacts and conversations with the SPIFFE/SPIRE team.

First, to conduct the security analysis over a concrete basis, the specialists targeted version `v0.6.1`. They defined the study's goals: undercover weaknesses in the core SPIFFE processes and to produce generic recommendations for the SPIFFE specification and all SPIFFE implementations. They aimed to document the expected security model for a SPIFFE implementation, show how SPIRE enforces (or fails to enforce) fundamental security properties, and identify gaps where further hardening can have a tangible impact.

Also, an essential part of the study was to define important non-goals. So the evaluators took the following perspective:

1. They assumed a rather generic deployment model that may not match every real-world deployment;
2. They made no judgment about how many times an attack will occur in a year but instead considered the relative probability and risk of different attacks (that is, the expected damage of various attacks);
3. And, they assumed that having a 100% secure system is impractical and focused on pragmatic changes with a reasonable cost/benefit ratio.

The evaluators enumerated the critical functions of a SPIFFE implementation, and as an outcome, they got the following critical concerns:

- Generation of workload identities based on workload attributes and system configuration;
- Attestation of workloads to make sure that workloads can access only identities to which they are entitled;
- Distribution of identities from the location where they are generated to the location where the workload can consume them;

- Distribution of trust bundles to workloads so that they can autonomously validate identities presented by other workloads without the need for an external service.

The participants of the analysis grouped attacks into categories based on supposed attacker goals. Each category is the inverse of a security goal for SPIFFE. All the attack types can be mounted against all components in SPIRE installation, namely, the server, the agent, and workloads. The participants considered that, depending on which component is the target, the attacks could happen with different difficulty levels and impacts on the system's security. The four defined attack types are:

- **Misrepresentation of identity**: type of attack that tricks the system into issuing identities that exploit weaknesses in the syntax or semantics of the validation of corresponding identity documents, for example, generating certificates with invalid characters or unverified identity attributes or trust bundles with rogue certificates.
- **Identity theft**: the purpose of this type of attack is to gain unauthorized access to the identity or critical material of another component. For example, a malicious workload may try to impersonate another workload, or a compromised SPIRE agent may try to impersonate the server.
- **Compromise**: in this type of attack, the attacker seeks to gain full control of one or more components. The victim component can be located on the same host or another host with network access. This attack type is intimately related to remote code execution (RCE).
- **Denial of service (DoS)**: these attacks disrupt system functionality by overloading or disabling one or more components, for example: A rogue entity can perform an SYN flood attack on the server; A compromised SPIRE agent can inundate the server with bogus certificate signing requests and prevent it from processing legitimate ones.

The security analysis participants also realized that an attacker might need specific capabilities to perform an attack of a particular type. For instance, to gain remote code execution, an attacker might require network access to a component with a buffer overflow vulnerability. In the security analysis report, it is clear that the listed capabilities do not correspond to known vulnerabilities but are based on the participants' knowledge of the SPIRE code and possible attack surface. The attacker capabilities identified are the following.

- **None**: a set of capabilities that are available by design. These are inherent in the design and deployment of SPIFFE or SPIRE.
- **Hammer**: the capability of asymmetrically generate a massive load on the victim component.
- **Escape**: the capability of escaping workload isolation boundaries within a node. One can use a kernel privilege escalation or other container escape vulnerability to escape these boundaries.
- **MitM**: the capability of intercept, eavesdrop, and possibly tamper with network communications between components.
- **PreAuthProto**: gain remote code execution or cause a crash by exploiting weaknesses in the pre-authenticated portion of the communication layer (e.g., TLS, Protobuf, gRPC).
- **X509Vuln**: gain remote code execution or cause a crash using a vulnerability in X.509 certificate parsing code.
- **CSRVuln**: gain remote code execution using a vulnerability in X.509 certificate signing request (CSR) parsing code.
- **CSROddity**: break the X.509 certificate signing request (CSR) parser to cause a crash, produce a malformed result, or trigger excessive resource consumption on the processing side (for example, by sending an arbitrarily large CSR or a large number of extensions).
- **MitigationBypass**: bypass a mitigating security control in some system component. This capability allowed the participants to estimate the risk of mitigated vulnerabilities if there is some chance that the mitigation is flawed.

With the attack types and capabilities to perform these attacks in hand, the security analysis proceeded to the next step. The participants assumed that attackers follow some logical steps to mount an attack. First, the attacker selects the type of attack he wants to mount based on the final goal. The second step is to select the victim component, i.e., the system's part that will be the attack target (server, agent, or workload). The third step is to select the attack origination point, which is one of the following: server, agent, workload, or an external entity. Finally, the attacker gains the capabilities required to mount the attack and initiate it.

The security analysis participants enumerated all the possible scenarios for attacks. For each scenario, they listed the type of attacks that can be mounted, the capabilities that these attacks require, and the security controls that are in place to mitigate them. They organized the results in a set of matrices, one for each attack type. The rows and columns identify the victims and origination points, respectively. The content of the cell describes the threats that are applicable in that particular scenario. These matrices were helpful to rank threats to a SPIRE deployment. The participants then estimate an attacker's likelihood of possessing each capability and assessing each attack's relative severity. Joining these two estimates, they produced a heuristically ranked list of potential risks to the system. This list gave a better understanding of the system's overall security properties and helped identify practical mitigations that could meaningfully improve its security. The security analysis can be found in the SPIRE's public repository[1].

5.2 Considered Approaches for the Security Analysis

To verify how the current SPIRE components behave under a more aggressive threat model, we established our base case to use regular attestation plugins for Kubernetes. The idea of this base case is to give us a notion of the security implications got by running the default plugins logic under such an aggressive threat model as the one typically assumed in confidential computing applications. In this sense, the two approaches analyzed were the following:

- Use of regular attestation plugins for Kubernetes (base case): this approach uses regular workload attestation plugins that work with the Kubernetes orchestrator to issue identities for confidential workloads. This approach is our base case. The base case help us understand the impact of the confidential workload's threat model in the SPIRE

[1]SPIRE code repository: https://github.com/spiffe/spire.

Agent's security analysis without modifications to issue identities for such sensitive workloads.

- SGX Helper: our proposed solution, described in 4.2.

As in the Figure 4.2, the Figure 5.1 depicts the trust levels of each component. The admin site is a hardened location, trusted-by-assumption. The untrusted portion is an environment controlled by third parties. The TEE represents the SCONE trust execution environment's trust boundaries.

This approach consists of the components and steps depicted in Figure 5.1. This figure shows the components involved in the SVID issuing process and a component named "other services," representing various components that the workload may want to communicate with after getting its identity.

The workflow to get an identity is as follows. In step 1, the operator posts the session with the initial configuration for the workload. In step 2, the operator registers the workload using the Workload API. To register the workload, the operator can use selectors that work with the k8s-related plugins types[2]. After creating the registration entry, the SPIRE Server provides the SPIRE Agent with the entry (assuming that the Agent has the parent ID defined in the registration entry) in step 3. In steps 4 and 5, the workload proves its identity to the SCONE Configuration and Attestation Service. Then, when the attestation process succeeds, the workload receives the initial configuration. In step 6, with the initial configuration in hand, the workload calls the Workload API to retrieve its identity. The SPIRE Agent handles the request and gets information from Kubernetes to build the set of selectors for the new request (step 7). In case of a successful SPIFFE attestation process, the Agent returns the SVID in step 8. The workload can then present the SVID in future message exchanges with other services (steps 9 and 10).

5.3 Security Analysis on the Base Case and Proposal

This section describes how we conducted the security analysis covering the base case and the proposed solution. The security analysis replications had two main objectives:

[2]https://github.com/spiffe/spire/blob/master/doc/plugin_agent_workloadattestor_k8s.md

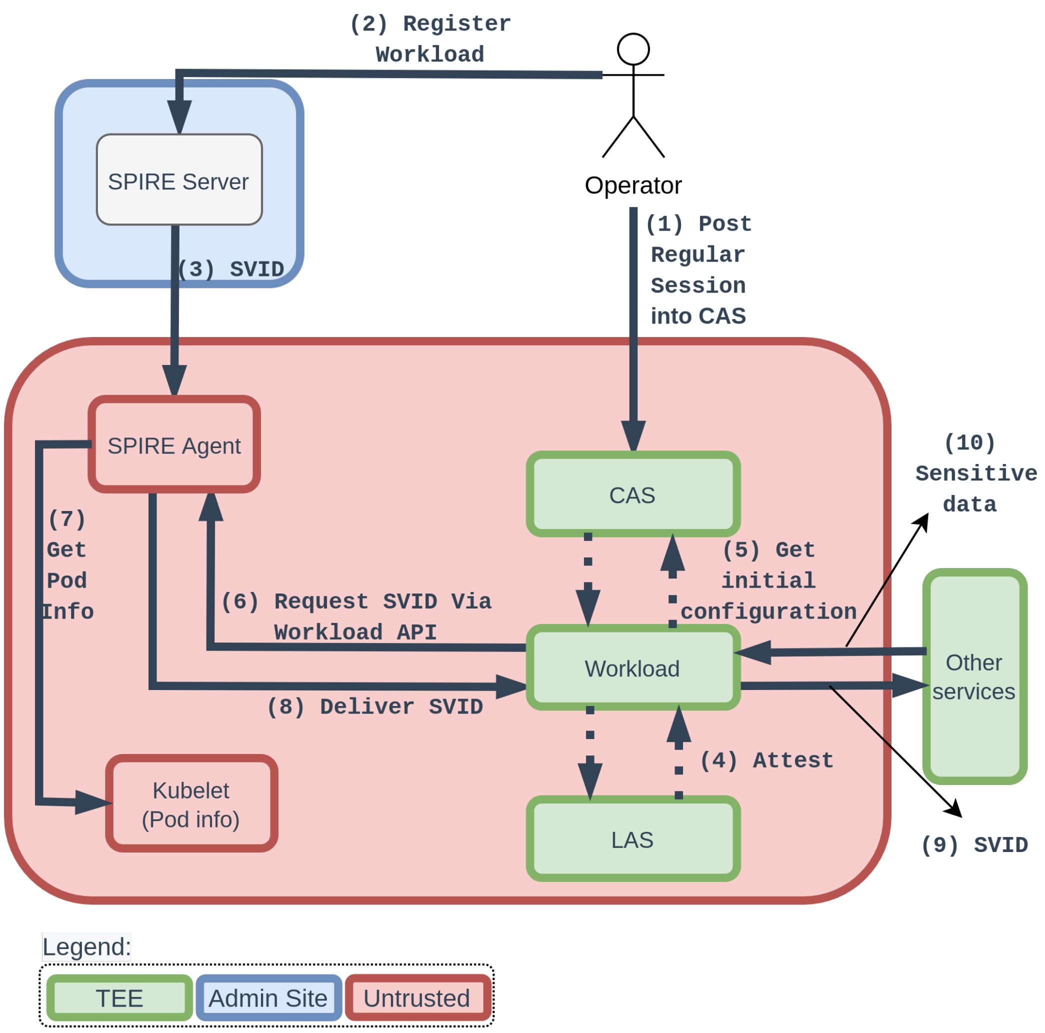


Figure 5.1: Identity issuing process workflow using the Kubernetes workload attestation plugin.

1. Understand how the base case behaves when it is subject to the confidential computing threat model;
2. Understand the strengths and weaknesses of both base case and our proposal, the SGX Helper.

Due to the social distancing needed because of the COVID-19 pandemics, we performed the security analysis activities either through virtual meetings or asynchronous work. We divided the replications into the following main phases:

1. Introduction to SPIFFE/SPIRE, its threat models, and attestation process (virtual meeting);
2. Introduction to the security analysis process and its output artifacts (virtual meeting);
3. Creation of new attack starting points and new capabilities derived from the confidential computing threat model (virtual meeting);
4. Analysis of each attack type filling the attack matrices (asynchronous tasks);
5. Estimation of impact scores (asynchronous tasks);
6. Summarization of the matrices and review (asynchronous tasks);

5.3.1 Phases 1 and 2

We invited five engineers with different levels of expertise on Intel SGX (and SCONE), software security, and confidential computing to perform our security analysis. Although they have different expertise levels, all the engineers researched and developed solutions for distributed systems' confidential computing.

The security assessment began with a presentation and discussions around the SPIFFE standard and the SPIRE tools[3]. The group of engineers had already been exposed to SPIFFE/SPIRE through other contacts, for example, participation in the SPIFFE's community Production Identity Day[4] and through other usages in the context of the larger research

[3]Set of slides available in `https://bit.ly/3cemCsn`.

[4]https://events.linuxfoundation.org/production-identity-day/. Last access in 2020-03-06.

project in which the current work was inserted. Once the group was sufficiently familiarized with the concepts, we explained the security assessment itself, explaining the SPIRE implementation's evaluation process. The presentations also covered the differences between the attestation process for both SPIRE and SGX.

5.3.2 Phase 3: Creation of New Attack Starting points and New Capabilities Derived from the Confidential Computing Threat Model

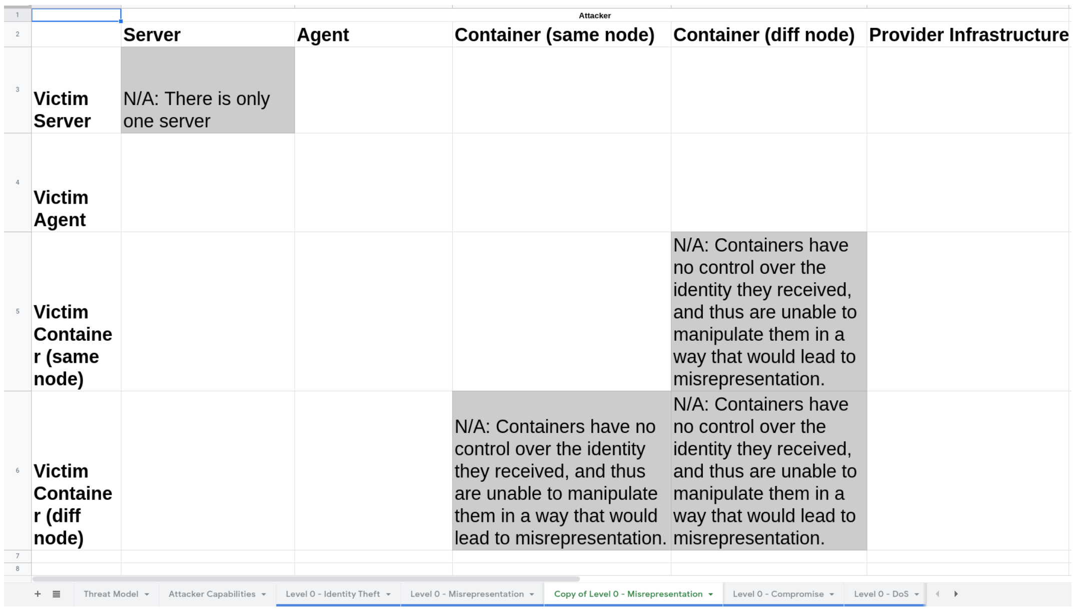
With the participants introduced to SPIRE, its attestation process, and its threat model, we started discussing what new capabilities an attacker would have considering the threat model for confidential computing. After the discussion, we added new items into the capabilities set presented in Section 5.1:

- **WMemory**: An attacker can write to any the memory region of arbitrary processes running in nodes of an untrusted infrastructure.
- **RMemory**: An attacker can read any memory region of arbitrary processes running in nodes of an untrusted infrastructure.
- **WDisk**: An attacker can write any location of the disks of arbitrary nodes of an untrusted infrastructure.
- **RDisk**: An attacker can read from any disks of arbitrary nodes of an untrusted infrastructure.
- **LimitResources**: An attacker, with the infrastructure provider perspective, can shutdown services, machines, or limit their execution.

Also, the group realized that there was a new starting point for attacks that would hold the new capabilities. The group then added a new starting point called **Provider Infrastructure**, representing the untrusted infrastructure of public clouds.

5.3.3 Phases 4, 5, and 6

With the updated capabilities set and the new attack starting point in hand, the group filled out the matrices asynchronously. Figure 5.2 shows an example of an attack matrix partially filled.



	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: There is only one server				
Victim Agent					
Victim Container (same node)				N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	
Victim Container (diff node)			N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	

Figure 5.2: Attack matrix sample: Misrepresentation.

We organized the results in tables, one for each pair “attack type” and “level”. We named our base case Level 0 (depicted in Figure 5.1). The scenario with the proposed solution is Level 1 (depicted in Figure 4.2). Although Level 1 has the SGX Helper instead of an Agent, the SGX Helper and SPIRE Agent play the same role. For simplicity and to maintain the tables uniform, all tables have entries called Agent and Victim Agent. Thus, for all effects, in the Level 1 tables, the entries labeled “Agent” and “Victim Agent” refer to the SGX Helper.

We summarized the matrices given by each participant into a matrix for each attack type joining the contents. We also summarized the scores. We colored the table cells according to the following conditions:

- Green represents a mitigated attack or otherwise not possible.
- Grey represents an attack that is not applicable.
- Red indicates that an attack is possible.
- Orange means that an attack may be possible under certain conditions.

5.4 Output Matrices

In this section, we present the output of our security analysis. We will discuss each attack type, compare the results for Level 0 and Level 1 tables, and highlight significant trade-offs.

5.4.1 Attack Matrix: Identity Theft

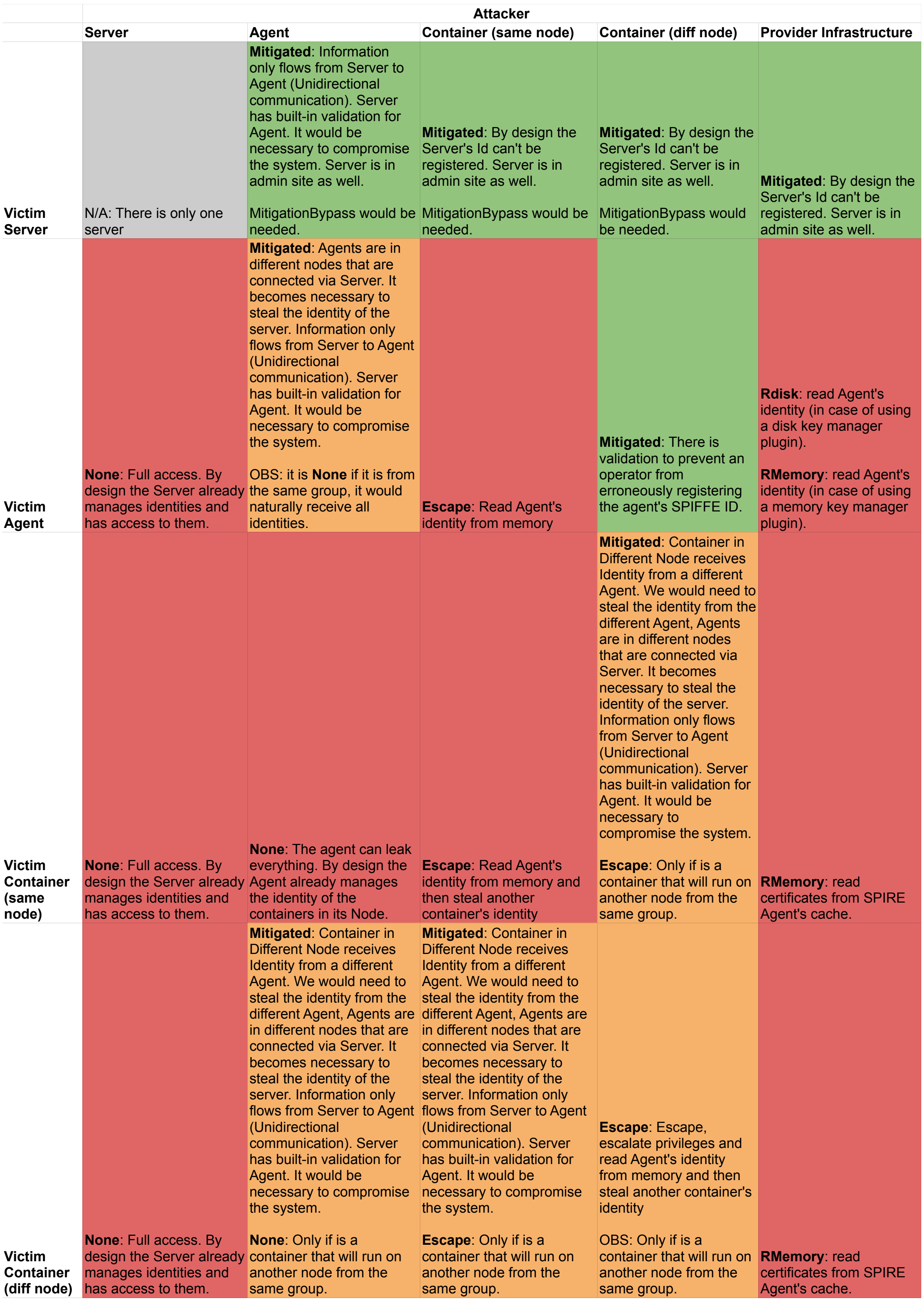
Figure 5.3 on the following page depicts the output table for identity theft in Level 0. When we assume that the attack starting point is the SPIRE Server, all the other components are defenseless. It occurs because the Server manages the identities for both workloads and Agents, which gives the Server power to impersonate any component in our analysis.

For the SPIRE Agent, the impact is less severe. The Agent can not fake its identity to the Server because of built-in validations. Moreover, the SVIDs go from the Server to the Agents. Similarly, such guarantees provided by the Server and SVID path from Server to Agents only prevent Agent-to-Agent attacks. In the case of Agent-to-Agent attacks, the evaluators identified that if a remote Agent belongs to the same group as the attacker's starting point (i. e., it has the same set of selectors), it will receive the same identities. However, when it comes to workload victims, the Agent may have the same potential as the Server. If the victim is a workload running in the compromised node or on a different node of the same group, an attack would be successful.

The participants evaluated the starting point as a workload running in a container in the same node (Agent as the reference point), the scenario is as severe as the Agent as the starting point. If the attacker got a container escape, it could steal SPIFFE artifacts from the Agent's memory. Furthermore, the issue with workloads in the same group persists. On the other hand, a workload running in a container in a different node can only use the same group artifice to steal identities.

The Provider Infrastructure starting point had a worrying evaluation. Considering the capabilities enumerated in the security analysis process, an attacker in this position has access to Agents' identities and the respective workloads under these agents' domains.

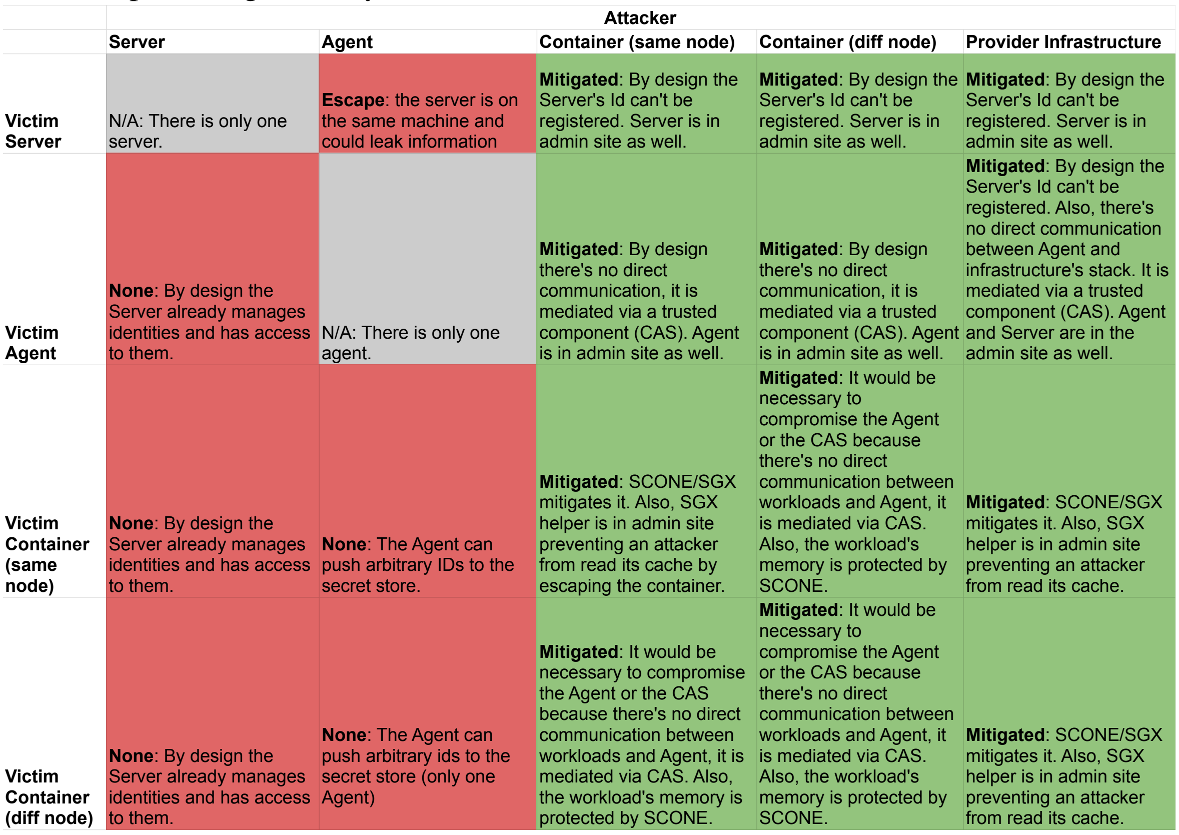
Figure 5.4 on page 44 shows the output table for Identity theft in Level 1. The colored visualization gives a good notion of our proposed solution. In the cases of Server and Agent as attacker starting points, the Level 0 threats remain. In Level 1, the workloads, even with



	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: There is only one server	**Mitigated**: Information only flows from Server to Agent (Unidirectional communication). Server has built-in validation for Agent. It would be necessary to compromise the system. Server is in admin site as well. MitigationBypass would be needed.	**Mitigated**: By design the Server's Id can't be registered. Server is in admin site as well. MitigationBypass would be needed.	**Mitigated**: By design the Server's Id can't be registered. Server is in admin site as well. MitigationBypass would be needed.	**Mitigated**: By design the Server's Id can't be registered. Server is in admin site as well.
Victim Agent	**None**: Full access. By design the Server already manages identities and has access to them.	**Mitigated**: Agents are in different nodes that are connected via Server. It becomes necessary to steal the identity of the server. Information only flows from Server to Agent (Unidirectional communication). Server has built-in validation for Agent. It would be necessary to compromise the system. OBS: it is **None** if it is from the same group, it would naturally receive all identities.	**Escape**: Read Agent's identity from memory	**Mitigated**: There is validation to prevent an operator from erroneously registering the agent's SPIFFE ID.	**Rdisk**: read Agent's identity (in case of using a disk key manager plugin). **RMemory**: read Agent's identity (in case of using a memory key manager plugin).
Victim Container (same node)	**None**: Full access. By design the Server already manages identities and has access to them.	**None**: The agent can leak everything. By design the Agent already manages the identity of the containers in its Node.	**Escape**: Read Agent's identity from memory and then steal another container's identity	**Mitigated**: Container in Different Node receives Identity from a different Agent. We would need to steal the identity from the different Agent, Agents are in different nodes that are connected via Server. It becomes necessary to steal the identity of the server. Information only flows from Server to Agent (Unidirectional communication). Server has built-in validation for Agent. It would be necessary to compromise the system. **Escape**: Only if is a container that will run on another node from the same group.	**RMemory**: read certificates from SPIRE Agent's cache.
Victim Container (diff node)	**None**: Full access. By design the Server already manages identities and has access to them.	**Mitigated**: Container in Different Node receives Identity from a different Agent. We would need to steal the identity from the different Agent, Agents are in different nodes that are connected via Server. It becomes necessary to steal the identity of the server. Information only flows from Server to Agent (Unidirectional communication). Server has built-in validation for Agent. It would be necessary to compromise the system. **None**: Only if is a container that will run on another node from the same group.	**Mitigated**: Container in Different Node receives Identity from a different Agent. We would need to steal the identity from the different Agent, Agents are in different nodes that are connected via Server. It becomes necessary to steal the identity of the server. Information only flows from Server to Agent (Unidirectional communication). Server has built-in validation for Agent. It would be necessary to compromise the system. **Escape**: Only if is a container that will run on another node from the same group.	**Escape**: Escape, escalate privileges and read Agent's identity from memory and then steal another container's identity OBS: Only if is a container that will run on another node from the same group.	**RMemory**: read certificates from SPIRE Agent's cache.

Figure 5.3: Attack matrix: Identity theft attack - Level 0.

escaping capabilities, can not read the memory of neighboring containers. Furthermore, the Agent is not in the same node as the containers, shielding the Agent's cache. The Agent placement and the protection provided by SGX mitigate the threats seen in Level 0 for the Provider Infrastructure starting point. The lack of direct communication of the Agent with untrusted parties significantly reduces the attack surface.



	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: There is only one server.	**Escape**: the server is on the same machine and could leak information	**Mitigated**: By design the Server's Id can't be registered. Server is in admin site as well.	**Mitigated**: By design the Server's Id can't be registered. Server is in admin site as well.	**Mitigated**: By design the Server's Id can't be registered. Server is in admin site as well.
Victim Agent	**None**: By design the Server already manages identities and has access to them.	N/A: There is only one agent.	**Mitigated**: By design there's no direct communication, it is mediated via a trusted component (CAS). Agent is in admin site as well.	**Mitigated**: By design there's no direct communication, it is mediated via a trusted component (CAS). Agent is in admin site as well.	**Mitigated**: By design the Server's Id can't be registered. Also, there's no direct communication between Agent and infrastructure's stack. It is mediated via a trusted component (CAS). Agent and Server are in the admin site as well.
Victim Container (same node)	**None**: By design the Server already manages identities and has access to them.	**None**: The Agent can push arbitrary IDs to the secret store.	**Mitigated**: SCONE/SGX mitigates it. Also, SGX helper is in admin site preventing an attacker from read its cache by escaping the container.	**Mitigated**: It would be necessary to compromise the Agent or the CAS because there's no direct communication between workloads and Agent, it is mediated via CAS. Also, the workload's memory is protected by SCONE.	**Mitigated**: SCONE/SGX mitigates it. Also, SGX helper is in admin site preventing an attacker from read its cache.
Victim Container (diff node)	**None**: By design the Server already manages identities and has access to them.	**None**: The Agent can push arbitrary ids to the secret store (only one Agent)	**Mitigated**: It would be necessary to compromise the Agent or the CAS because there's no direct communication between workloads and Agent, it is mediated via CAS. Also, the workload's memory is protected by SCONE.	**Mitigated**: It would be necessary to compromise the Agent or the CAS because there's no direct communication between workloads and Agent, it is mediated via CAS. Also, the workload's memory is protected by SCONE.	**Mitigated**: SCONE/SGX mitigates it. Also, SGX helper is in admin site preventing an attacker from read its cache.

Figure 5.4: Attack matrix: Identity theft attack - Level 1.

5.4.2 Attack Matrix: Misrepresentation

Figure 5.5 on page 46 shows the output table for misrepresentation attack in Level 0. From the perspective of the SPIRE Server as the starting point, we obtained a severe threat scenario again. Since the Server controls the SVID signing keys, it is in a position to mint arbitrary identities.

With the Agent as the starting point, attacks against Server and other Agent victims are mitigated. In these cases, the Server is sending identities, so the Agent can not trick the system by exploiting weaknesses in the document validations. However, when the workload in the same node is the victim, a malicious Agent has more power and can induce the workload to trust in unauthorized SVIDs. In addition, there is a risk of asking the Server to sign CSRs with odd properties crafted to exploit victims. This last possibility applies especially to workload containers running in other nodes if the attacker could gain the MitM capability.

The mitigation for attacks against Server and Agent is still in place for the origin points on workload containers running in the same node and other nodes. In general, for workloads-to-workloads attacks, the containers have no control over the identity they receive and thus cannot manipulate them in a way that would lead to misrepresentation. There is only one exception for attackers running in the same node: after an escape, an attacker could pull the local clock back, tricking neighboring containers into accepting expired identities.

A starting point at the provider infrastructure could trick the Agent and the workload containers for the misrepresentation attack. An attack against the Server is mitigated because of its placement.

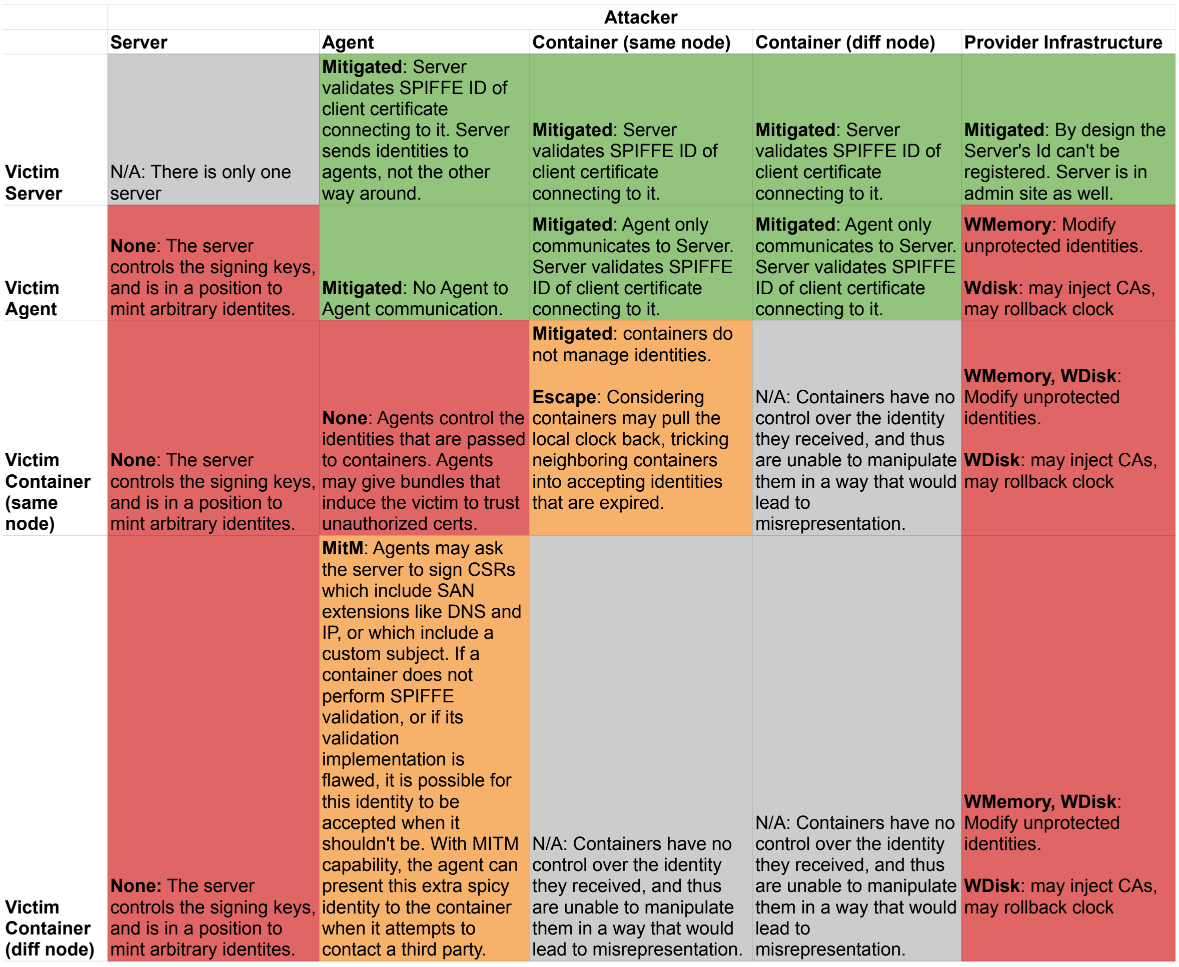
Moving to Level 1, Figure 5.4 on the previous page, shows that our proposed solution does not impact the analysis of the Server as the attack origin point. However, analyzing the Agent as the starting point, the result was more severe than in Level 0 because, in Level 1, the Agent does not need any particular condition to attack victim workloads in any node.

In the case of workload to workload attacks in the same node, the analysis remains the same, considering that the current implementation of SCONE has no support of a trusted clock. It is also an open issue of other approaches to run applications inside SGX.

In opposition to Level 0, in Level 1, the TEE mitigates the attacks against the workloads because of the memory protection associated with the attestation process performed by CAS before delivering identities. Also, attacks against the Server and the Agent are mitigated because these components are in the Admin Site. The evaluation revealed that the Agent's placement is critical since it can be used as a foothold to perform attacks against workloads, even if they run inside SGX. In other words, the Agent has sensitive SPIFFE artifacts in plain text as its data-in-use.

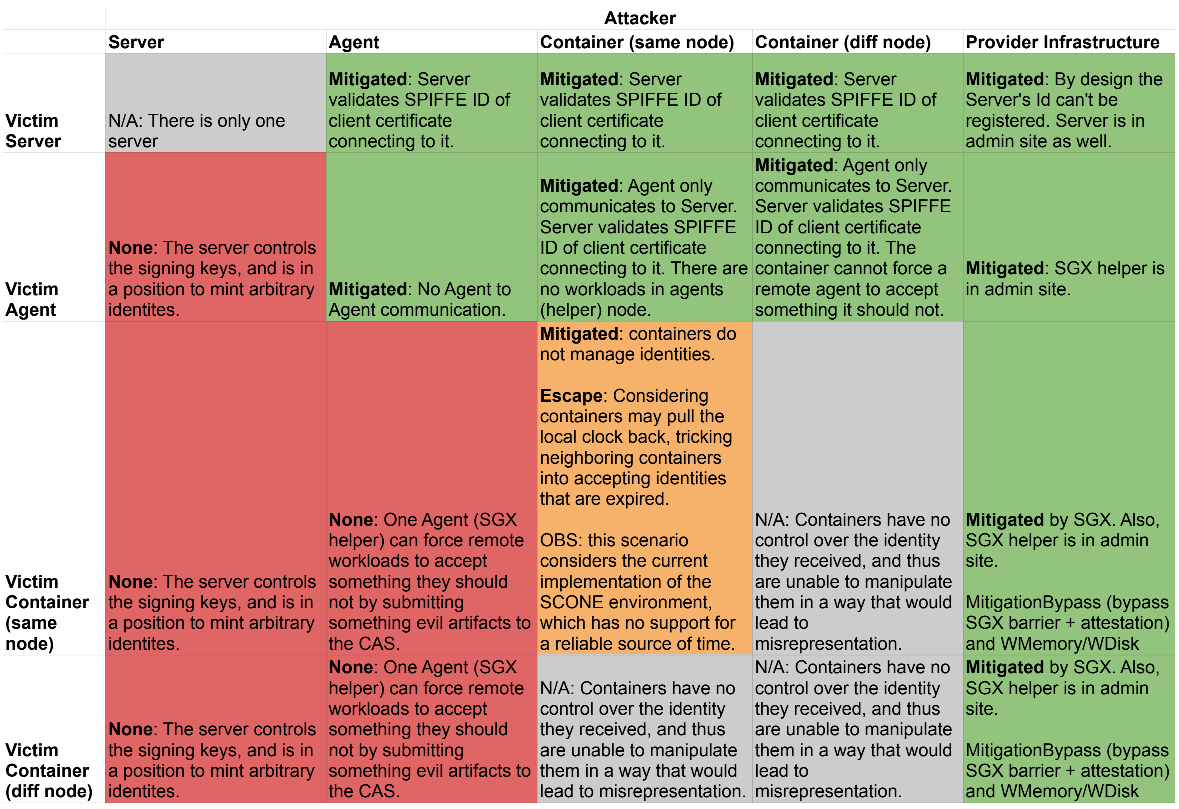
5.4.3 Attack matrix: Compromise

Figure 5.7 on page 48 shows the output table for the compromise attack in Level 0. In this type of attack, we mostly considered a set of capabilities that lead to remote code execution,



	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: There is only one server	**Mitigated**: Server validates SPIFFE ID of client certificate connecting to it. Server sends identities to agents, not the other way around.	**Mitigated**: Server validates SPIFFE ID of client certificate connecting to it.	**Mitigated**: Server validates SPIFFE ID of client certificate connecting to it.	**Mitigated**: By design the Server's Id can't be registered. Server is in admin site as well.
Victim Agent	**None**: The server controls the signing keys, and is in a position to mint arbitrary identites.	**Mitigated**: No Agent to Agent communication.	**Mitigated**: Agent only communicates to Server. Server validates SPIFFE ID of client certificate connecting to it.	**Mitigated**: Agent only communicates to Server. Server validates SPIFFE ID of client certificate connecting to it.	**WMemory**: Modify unprotected identities. **Wdisk**: may inject CAs, may rollback clock
Victim Container (same node)	**None**: The server controls the signing keys, and is in a position to mint arbitrary identites.	**None**: Agents control the identities that are passed to containers. Agents may give bundles that induce the victim to trust unauthorized certs.	**Mitigated**: containers do not manage identities. **Escape**: Considering containers may pull the local clock back, tricking neighboring containers into accepting identities that are expired.	N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	**WMemory, WDisk**: Modify unprotected identities. **WDisk**: may inject CAs, may rollback clock
Victim Container (diff node)	**None:** The server controls the signing keys, and is in a position to mint arbitrary identites.	**MitM**: Agents may ask the server to sign CSRs which include SAN extensions like DNS and IP, or which include a custom subject. If a container does not perform SPIFFE validation, or if its validation implementation is flawed, it is possible for this identity to be accepted when it shouldn't be. With MITM capability, the agent can present this extra spicy identity to the container when it attempts to contact a third party.	N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	**WMemory, WDisk**: Modify unprotected identities. **WDisk**: may inject CAs, may rollback clock

Figure 5.5: Attack matrix: Misrepresentation attack - Level 0.



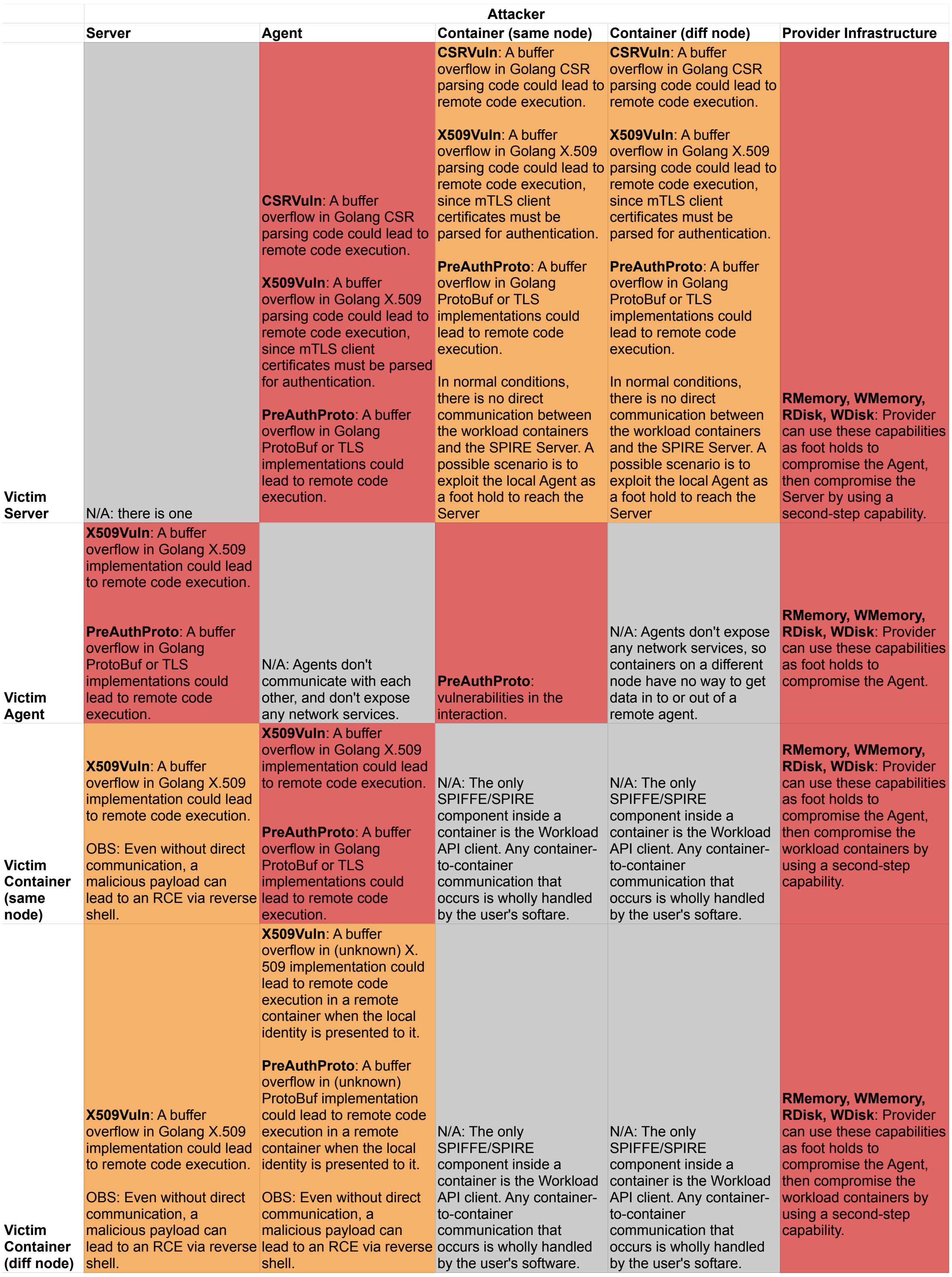
	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: There is only one server	**Mitigated**: Server validates SPIFFE ID of client certificate connecting to it.	**Mitigated**: Server validates SPIFFE ID of client certificate connecting to it.	**Mitigated**: Server validates SPIFFE ID of client certificate connecting to it.	**Mitigated**: By design the Server's Id can't be registered. Server is in admin site as well.
Victim Agent	**None**: The server controls the signing keys, and is in a position to mint arbitrary identites.	**Mitigated**: No Agent to Agent communication.	**Mitigated**: Agent only communicates to Server. Server validates SPIFFE ID of client certificate connecting to it. There are no workloads in agents (helper) node.	**Mitigated**: Agent only communicates to Server. Server validates SPIFFE ID of client certificate connecting to it. The container cannot force a remote agent to accept something it should not.	**Mitigated**: SGX helper is in admin site.
Victim Container (same node)	**None**: The server controls the signing keys, and is in a position to mint arbitrary identites.	**None**: One Agent (SGX helper) can force remote workloads to accept something they should not by submitting something evil artifacts to the CAS.	**Mitigated**: containers do not manage identities. **Escape**: Considering containers may pull the local clock back, tricking neighboring containers into accepting identities that are expired. OBS: this scenario considers the current implementation of the SCONE environment, which has no support for a reliable source of time.	N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	**Mitigated** by SGX. Also, SGX helper is in admin site. MitigationBypass (bypass SGX barrier + attestation) and WMemory/WDisk
Victim Container (diff node)	**None**: The server controls the signing keys, and is in a position to mint arbitrary identites.	**None**: One Agent (SGX helper) can force remote workloads to accept something they should not by submitting something evil artifacts to the CAS.	N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	N/A: Containers have no control over the identity they received, and thus are unable to manipulate them in a way that would lead to misrepresentation.	**Mitigated** by SGX. Also, SGX helper is in admin site. MitigationBypass (bypass SGX barrier + attestation) and WMemory/WDisk

Figure 5.6: Attack matrix: Misrepresentation attack - Level 1.

enabling an attacker to get full control over the victim components as defined in the attack definition. To do so, we consider that even SGX-enabled workloads are vulnerable to binary exploitation. From the basics such as simple buffer overflows overwriting stack pointers to more complex return-oriented programming attacks such as the presented by Schwarz et al [Schwarz, Weiser e Gruss 2019]. Nonetheless, we considered that due to specificities of the SCONE platform implementation, the changes in the behavior of exploits when switching from Level 0 to Level 1 could be subject of a deeper analysis. However, it is out of the scope of this work.

Using the SPIRE Server as the starting point, an attacker could take advantage of capabilities to exploit both vulnerabilities in the X.509 implementation and vulnerabilities in the protocol buffers implementation (X509Vuln and PreAuthProto capabilities). The analysis output also considered that under certain conditions, the Server could compromise the workload containers. The Server could exploit the workload containers via indirect communication injecting the exploit payload in the SVIDs, using the X509Vuln capability.

Analyzing the Agent as the attack starting point, the same capabilities used by a Server



	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: there is one	**CSRVuln**: A buffer overflow in Golang CSR parsing code could lead to remote code execution. **X509Vuln**: A buffer overflow in Golang X.509 parsing code could lead to remote code execution, since mTLS client certificates must be parsed for authentication. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could lead to remote code execution.	**CSRVuln**: A buffer overflow in Golang CSR parsing code could lead to remote code execution. **X509Vuln**: A buffer overflow in Golang X.509 parsing code could lead to remote code execution, since mTLS client certificates must be parsed for authentication. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could lead to remote code execution. In normal conditions, there is no direct communication between the workload containers and the SPIRE Server. A possible scenario is to exploit the local Agent as a foot hold to reach the Server	**CSRVuln**: A buffer overflow in Golang CSR parsing code could lead to remote code execution. **X509Vuln**: A buffer overflow in Golang X.509 parsing code could lead to remote code execution, since mTLS client certificates must be parsed for authentication. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could lead to remote code execution. In normal conditions, there is no direct communication between the workload containers and the SPIRE Server. A possible scenario is to exploit the local Agent as a foot hold to reach the Server	**RMemory, WMemory, RDisk, WDisk**: Provider can use these capabilities as foot holds to compromise the Agent, then compromise the Server by using a second-step capability.
Victim Agent	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could lead to remote code execution.	N/A: Agents don't communicate with each other, and don't expose any network services.	**PreAuthProto**: vulnerabilities in the interaction.	N/A: Agents don't expose any network services, so containers on a different node have no way to get data in to or out of a remote agent.	**RMemory, WMemory, RDisk, WDisk**: Provider can use these capabilities as foot holds to compromise the Agent.
Victim Container (same node)	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. OBS: Even without direct communication, a malicious payload can lead to an RCE via reverse shell.	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could lead to remote code execution.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's softare.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's softare.	**RMemory, WMemory, RDisk, WDisk**: Provider can use these capabilities as foot holds to compromise the Agent, then compromise the workload containers by using a second-step capability.
Victim Container (diff node)	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. OBS: Even without direct communication, a malicious payload can lead to an RCE via reverse shell.	**X509Vuln**: A buffer overflow in (unknown) X. 509 implementation could lead to remote code execution in a remote container when the local identity is presented to it. **PreAuthProto**: A buffer overflow in (unknown) ProtoBuf implementation could lead to remote code execution in a remote container when the local identity is presented to it. OBS: Even without direct communication, a malicious payload can lead to an RCE via reverse shell.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's softare.	**RMemory, WMemory, RDisk, WDisk**: Provider can use these capabilities as foot holds to compromise the Agent, then compromise the workload containers by using a second-step capability.

Figure 5.7: Attack Matrix: Compromise - Level 0.

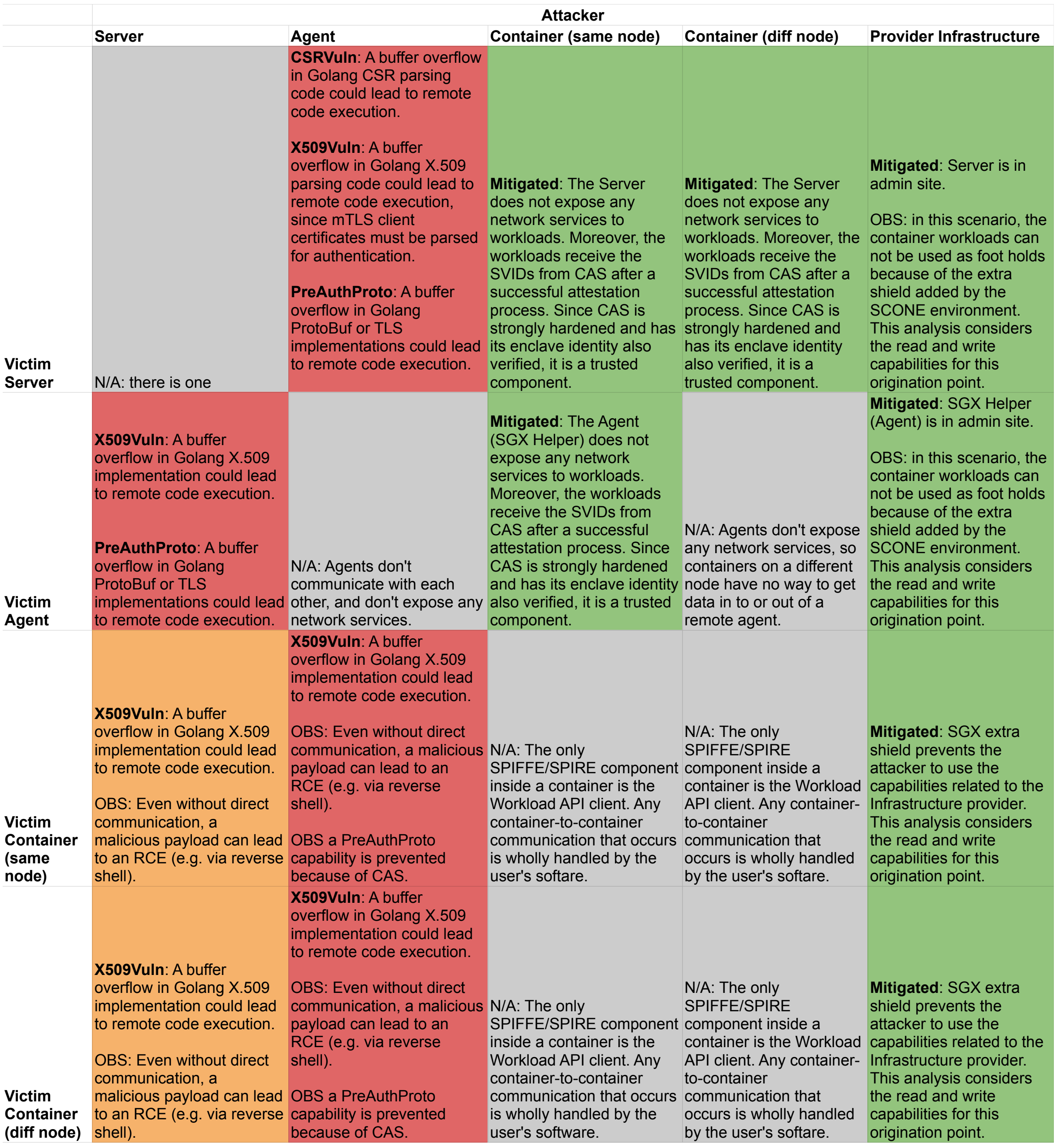
origin point are helpful to an attacker. Additionally, as the Agent sends CSRs so the Server can sign the SVIDs for that Agent, CSRVuln also is used to compromise the Server. However, the analysis did not consider Agent to Agent attacks because there is no lateral communication between Agents in a regular SPIRE deployment. In this sense, it is essential to note that specific deployments could have a (unknown) combination of plugins that leads to lateral communication. The workloads containers in the same node are exploitable via the X509Vuln capability and PreAuthProto because the Agent exposes the Workload API to the workloads.

For the workload containers as attack origin points, as the Agent is in the same node, the analysis output pointed the Agent as a foothold to get to the Server one more time. Similar to the Agent to workload attack, the Workload API exposed by the Agent is an attack vector to exploit with the PreAuthProto capability.

Still, the analysis for the Provider Infrastructure as the attack origin point showed that the Agent is, again, a potential foothold to compromise the Server, this time considering the particular capabilities in the Provider Infrastructure hands. For all the other victim types, the Provider Infrastructure capabilities are sufficient to exploit the victims. In the case of workload containers, despite the SGX shielding, the Agent can be used as a foothold to compromise them using a second-step capability.

Figure 5.8 on the next page shows the analysis for Level 1. For Server and Agent as starting points for attacks, the analysis is similar to Level 0. Nevertheless, the results for other starting points are promising. In general, the placement of Server and Agent in Level 1 extinguishes the attack vectors.

For workload containers as attack origin points, there is no local Agent to use as a foothold to exploit the Server. In addition, there is no Workload API to abuse. The same happens with the Provider Infrastructure as the origin point. The Agent does not provide any exploitable API and, with no foothold, the attacks against the SPIRE Server are mitigated likewise. Finally, the SCONE-SGX security properties protect the workload containers against the capabilities of the Provider Infrastructure (WMemory, RMemory, WDisk, RDisk).



	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: there is one	**CSRVuln**: A buffer overflow in Golang CSR parsing code could lead to remote code execution. **X509Vuln**: A buffer overflow in Golang X.509 parsing code could lead to remote code execution, since mTLS client certificates must be parsed for authentication. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could lead to remote code execution.	**Mitigated**: The Server does not expose any network services to workloads. Moreover, the workloads receive the SVIDs from CAS after a successful attestation process. Since CAS is strongly hardened and has its enclave identity also verified, it is a trusted component.	**Mitigated**: The Server does not expose any network services to workloads. Moreover, the workloads receive the SVIDs from CAS after a successful attestation process. Since CAS is strongly hardened and has its enclave identity also verified, it is a trusted component.	**Mitigated**: Server is in admin site. OBS: in this scenario, the container workloads can not be used as foot holds because of the extra shield added by the SCONE environment. This analysis considers the read and write capabilities for this origination point.
Victim Agent	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could lead to remote code execution.	N/A: Agents don't communicate with each other, and don't expose any network services.	**Mitigated**: The Agent (SGX Helper) does not expose any network services to workloads. Moreover, the workloads receive the SVIDs from CAS after a successful attestation process. Since CAS is strongly hardened and has its enclave identity also verified, it is a trusted component.	N/A: Agents don't expose any network services, so containers on a different node have no way to get data in to or out of a remote agent.	**Mitigated**: SGX Helper (Agent) is in admin site. OBS: in this scenario, the container workloads can not be used as foot holds because of the extra shield added by the SCONE environment. This analysis considers the read and write capabilities for this origination point.
Victim Container (same node)	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. OBS: Even without direct communication, a malicious payload can lead to an RCE (e.g. via reverse shell).	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. OBS: Even without direct communication, a malicious payload can lead to an RCE (e.g. via reverse shell). OBS a PreAuthProto capability is prevented because of CAS.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's softare.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's softare.	**Mitigated**: SGX extra shield prevents the attacker to use the capabilities related to the Infrastructure provider. This analysis considers the read and write capabilities for this origination point.
Victim Container (diff node)	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. OBS: Even without direct communication, a malicious payload can lead to an RCE (e.g. via reverse shell).	**X509Vuln**: A buffer overflow in Golang X.509 implementation could lead to remote code execution. OBS: Even without direct communication, a malicious payload can lead to an RCE (e.g. via reverse shell). OBS a PreAuthProto capability is prevented because of CAS.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's softare.	**Mitigated**: SGX extra shield prevents the attacker to use the capabilities related to the Infrastructure provider. This analysis considers the read and write capabilities for this origination point.

Figure 5.8: Attack matrix: Compromise - Level 1.

5.4.4 Attack Matrix: DoS

Figure 5.9 on the following page shows the output table for denial of service attack in Level 0. Since the Server has full power over the identities, it can deny them to the Agents and workloads or overload these victims.

If the SPIRE Agent is the origin point for a DoS attack, it can overload the Server with numerous CSRs or attempt a more complex attack. Typical binary vulnerabilities like overflows may crash the Server, preventing it from serving other Agents. Also, an attacker can use the CSRoddity capability to issue CSRs that could lead the Server to consume all its resources.

The analysis for workload containers as the attacker starting point showed that a Server victim is vulnerable using the Agent as a foothold. Once the Agent is compromised, the attacker uses it with the same capabilities in the cell showing the Agent as the starting point. Workload containers also can spoil the Agent's service. However, workloads from different nodes can only affect Agents using the MitM capability.

If the starting point for the attacker is the Provider Infrastructure, the attacker can use the Agent as a foothold (similar to the last scenario). Besides, in this scenario, the attacker has the powerful capability to limit resources for the applications. In this sense, a DoS coming from the Provider Infrastructure attacker is trivial.

In comparison with Level 0, Level 1 has clear advantages. Although the workload situation is the same, an attack starting from the Provider Infrastructure has no importance against both the Server and Agent since the Agent and Server are in a different location and do not expose network services. Furthermore, as the Agent in Level 1 does not expose the workload API to workloads, the attacks against the Agent are mitigated.

Level 1, which has the analysis depicted in Figure 5.10 on page 53, also mitigates DoS attacks coming from the workloads against the Server. The Server does not expose network services to components other than Agents. The workloads receive the SVIDs posted into CAS after a successful attestation process. In SCONE workloads, this process is triggered only at start-up time.

	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: There is one	**CSRVuln**: A buffer overflow in Golang CSR parsing code could crash the server. **CSRoddity:** Exceptionally large, never-ending, or other odd CSR qualities could lead the server to consume all its resources. **X509Vuln:** A buffer overflow in Golang X.509 parsing code could crash the server, since mTLS client certificates must be parsed for authentication. **PreAuthProto:** A buffer overflow in Golang ProtoBuf or TLS implementations could crash the server. **Hammer:** An agent could send a large number of CSRs to the server, forcing it to consume computational resources.	**X509Vuln**: A buffer overflow in Golang X.509 Agent's parsing code could allow the workloads to use the Agents as foot holds exploit the Server. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations used by the Agent could allow the workloads to use the Agents as foot holds exploit the Server. In normal conditions, there is no direct communication between the workload containers and the SPIRE Server. A possible scenario is to exploit the local Agent as a foot hold to reach the Server.	**X509Vuln**: A buffer overflow in Golang X. 509 Agent's parsing code could allow the workloads to use the Agents as foot holds exploit the Server. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations used by the Agent could allow the workloads to use the Agents as foot holds exploit the Server. In normal conditions, there is no direct communication between the workload containers and the SPIRE Server. A possible scenario is to exploit the local Agent as a foot hold to reach the Server.	**RMemory, WMemory, RDisk, WDisk**: Provider can use these capabilities as foot holds to poison Agent data, leading to a second-step capability to exploit the Server.
Victim Agent	**None**: Can deny identities or overload the agent.	**MitM**: Layer 2 attacks like ARP poisoning make it possible to block agent-to-server communication. Access to the network enables interference with other agent-to-server communication (ARP, flood)	**Escape**: By getting access to the network could interfere with other agent-to-server communication **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could crash the server. **Hammer**: An agent could send a large number of CSRs to the server, forcing it to consume computational resources.	**MitM**: Layer 2 attacks like flooding make it possible to block agent-to-server communication	**LimitResouces**: Provider can shutdown the services/machines or limit their execution.
Victim Container (same node)	**None**: Can deny identities or overload the agents and workloads.	**None**: The agent issues identity to local containers, and is in a position to omit or deny the issuance of identities.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	**LimitResouces**: Provider can shutdown the services/machines or limit their execution.
Victim Container (diff node)	**None**: Can deny identities or overload the agents and workloads.	N/A: Agents don't provide services to containers on other nodes, and are therefor not in a position to tamper with SPIFFE/SPIRE functioning of remote containers. Furthermore, they don't have the ability to write data to the server, preventing them from modifying registration data or otherwise affecting identity issuance to remote containers.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	**LimitResouces**: Provider can shutdown the services/machines or limit their execution.

Figure 5.9: Attack matrix: DoS attack - Level 0.

	Attacker				
	Server	**Agent**	**Container (same node)**	**Container (diff node)**	**Provider Infrastructure**
Victim Server	N/A: There is one	**CSRVuln**: A buffer overflow in Golang CSR parsing code could crash the server. **CSRoddity**: Exceptionally large, never-ending, or other odd CSR qualities could lead the server to consume all its resources. **X509Vuln**: A buffer overflow in Golang X.509 parsing code could crash the server, since mTLS client certificates must be parsed for authentication. **PreAuthProto**: A buffer overflow in Golang ProtoBuf or TLS implementations could crash the server. **Hammer**: An agent could send a large number of CSRs to the server, forcing it to consume computational resources.	**Mitigated**: The Server does not expose any network services to workloads. Moreover, the workloads receive the SVIDs from CAS after a successful attestation process. In this sense, the SCONE-enabled workloads only trigger the attestation process in the start-up.	**Mitigated**: The Server does not expose any network services to workloads. Moreover, the workloads receive the SVIDs from CAS after a successful attestation process. In this sense, the SCONE-enabled workloads only trigger the attestation process in the start-up.	N/A: The Server does not expose any network services to the infrastructure where the workloads are deployed on.
Victim Agent	**None**: Can deny identities or overload the agent.	**MitM**: Layer 2 attacks like ARP poisoning make it possible to block agent-to-server communication. Access to the network enables interference with other agent-to-server communication (ARP, flood)	**Mitigated**: By design there's no direct communication, it is mediated via a trusted component (CAS). Agent is in admin site as well.	**Mitigated**: By design there's no direct communication, it is mediated via a trusted component (CAS). Agent is in admin site as well.	N/A: The Agent does not expose any network services to the infrastructure where the workloads are deployed on.
Victim Container (same node)	**None**: Can deny identities or overload the agents and workloads.	**None**: The agent issues identity to the container workloads, and is in a position to omit or deny the issuance of identities.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	**LimitResouces**: Provider can shutdown the services/machines or limit their execution.
Victim Container (diff node)	**None**: Can deny identities or overload the agents and workloads.	N/A: Agents don't provide services to containers on other nodes, and are therefor not in a position to tamper with SPIFFE/SPIRE functioning of remote containers. Furthermore, they don't have the ability to write data to the server, preventing them from modifying registration data or otherwise affecting identity issuance to remote containers.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	N/A: The only SPIFFE/SPIRE component inside a container is the Workload API client. Any container-to-container communication that occurs is wholly handled by the user's software.	**LimitResouces**: Provider can shutdown the services/machines or limit their execution.

Figure 5.10: Attack matrix: DoS attack - Level 1.

5.5 Conclusions

In the context of the security analysis, the solution proposed in this work using the SGX Helper was considered effective against attacks from the Infrastructure Provider starting point, except for DoS attacks, according to the participants. The solution presented better results against the Identity Theft attack type, mitigating all attacks from workload containers.

The security analysis showed that the components' placement was crucial to mitigate various attack combinations, mainly by decoupling the communication between Agents and workloads via a hardened and trusted component, the CAS.

However, putting the Agent in the same machine as the Server is risky. It relaxes the boundary between the Agent and the Server, and a compromised Agent can leak Server's sensitive information. Putting the Agent and the Server inside fully isolated enclaves may enforce the Agent-Server trust boundary.

5.6 Challenges in the Security Analysis Process

We faced challenges throughout the security analysis process. The security analysis was a long-running process, and because of that, participants eventually became unavailable. Consequently, sometimes only four of five participants contributed to the final result in some tables. Also, it affected discussions to have a common understanding, once one participant did not respond to further conversations after filling all tables.

With our process, we learned that the longer the interval between security analysis discussions, the more difficult it is for participants to resume the analysis and remember the reasons for certain decisions. To avoid problems with participants' availability and productivity, we recommend those interested in running further security analysis to try to shorten the intervals between the time slices dedicated to filling attack tables.

Another challenge involved in the security analysis is the identification of misunderstandings about mitigated attacks. Let us describe an example to illustrate this challenge.

In the analysis for Identity Theft in Level 0, one participant initially established workload-to-workload attacks as mitigated because of SCONE-SGX. The premise was that if the victim workload runs inside SCONE, the memory protection prevents the attack from coming from another workload that escaped its container. Although the condition is true, the participant ignored the SPIRE Agent running in the same node. The analysis can not ignore the Agent because, in Level 0, the Agent is a weak component in the system. It caches the SPIRE registration entries in memory, exposing this information to potential attackers. With the escape capability and privilege escalation, the entries are no longer secrets.

Similarly, the analysis must consider attacks using remote code execution to compromise workload containers, even whether the workloads run inside enclaves. An attack using modern binary exploitation techniques may be more complex, with more variables for an attacker to control, but still feasible. Schwarz et al. [Schwarz, Weiser e Gruss 2019] demonstrated it by applying the return-oriented programming technique in SGX enclaves to execute actions outside the enclave.

We verified that differences in understanding could lead to different results. Some analysis results may be imprecise, similar to the described previously. These cases must be identified and moderated. However, most of the differences found in the security analysis were essential to the bigger picture. In this sense, the diverse ways of thinking about the system help to cover more possibilities. Merging the individual results is a challenging task, but the benefits of diversity worth it to the final results.

Chapter 6

Performance Experiments

In this chapter, we present an evaluation of the proposed solution when it comes to the SVID delivery time for confidential workloads and container image build times.

6.1 Issuing Identities for Confidential Workloads

For each security layer added to a computing system, there is an additional cost of usability or performance. In the case of the integration between confidential workloads and SPIRE, we considered that the critical parts of the workload's workflow are the startup time and the time needed for a workload to get an SVID. The aspects of the usability of the SPIRE part of our solution remain unchanged. Also, on the SCONE side, the usage experiment is similar to any other confidential workload operation. In other words, assuming an operator familiar with both SCONE and SPIRE environments, there are no workflow changes with the operation of workloads using the integration between SPIRE and confidential workloads. Then we designed an experiment to check if there is a significant difference with respect to performance aspects (and how big this difference is) between the two scenarios: confidential workloads and regular workloads, both orchestrated by Kubernetes.

The following research questions guided the experiments:

- **RQ1**: Is the difference between the startup times of confidential workloads and regular workloads significant?
- **RQ2**: Is the difference between the SVID delivery time for confidential workloads and

Node	**RAM Size**	**Number of vCPUs**
K8s master	2 GB	2
K8s worker	8 GB	4

Table 6.1: VM specifications for experiment execution.

regular workloads significant?

We chose a complete factorial design, and we considered the workload configuration, associated with the use or not of SGX, as a factor. This factor is named scenario and has three levels:

- Regular Workload: a regular workload written in Python3 uses the SPIRE Agent's workload API to get an SVID.
- Confidential Workload: a confidential workload written in Python3 uses the SGX Helper to get an SVID.
- Confidential Workload with FSPF: a workload written in Python3 uses the SGX Helper to get an SVID. We used the SCONE FSPF feature to deploy this scenario, which means that, in this scenario, the application code is confidential, and the key to decrypt the code is delivered by CAS after a successful attestation process.

We decided to distinguish the second and the third levels because even confidential workloads may have different security requirements. For workloads that do not need to check on interpreted code or dependencies (like in case of static-compiled binaries), the second level may fit well. For workloads that need more guarantees on dependencies, interpreted code, or specific files in the container images, the third level is the best option.

6.1.1 Environment and Tools

To run this experiment, we set up a Kubernetes cluster with two VMs. One VM for the Kubernetes master node and the other VM for the Kubernetes worker. Table 6.1.1 shows each VM's specification.

We placed the experiment script in the worker machine and used the `kubectl` command-line application to deploy the workloads. We measured the time in three different phases of the workload execution. The first one ($T1$) captured the time of workload creation on Kubernetes (using the `kubectl` tool). The time $T2$ refers to the time of the container entry point execution. All the container entry points used in this experiment have a single concern: record the time when the container becomes ready. After $T2$, we captured the time $T3$ when the workload got an SVID either from the Workload API or from the CAS. We used Python3 time libraries and the GNU core utilities to measure $T1$, $T2$, and $T3$.

6.1.2 Methodology

To answer the research questions RQ1 and RQ2, presented in section 6.1, we executed 5000 replicas of each treatment. Before each replica the environment was reset, ensuring no interference between the replicas. After the experiment execution, we parsed our data into two metrics. The first one is equal to $T2 - T1$, called start-up time (time needed for the container to be ready). The second one is set to $T3 - T2$, the time spent to get an SVID (SVID delivery time) after the container entry point is executed. With these data in hand, we analyzed the difference between the times collected for each scenario. To do this, we generated bootstrap estimates for the difference of the medians of each scenario using bias-corrected and accelerated (BCa) bootstrap intervals to compare the scenarios. We chose BCa because it corrects for bias and skewness in the distribution of bootstrap estimates. According to Greenwood [Greenwood 2014], if the interval on the difference between two arbitrary *A* and *B* does not contain zero, then we can say that zero probably is not in the true values at the selected confidence interval. Thus, we should reject the claim that *A* and *B* are equal.

6.1.3 Discussion and Conclusions

In Figure 6.1, we can see the distribution of the time needed to execute the container entry point (Container readiness).

The distribution of the time spent to get an SVID after the container is ready is illustrated in Figure 6.2. There are some outliers. The distribution is not symmetric. The scenario with confidential workloads with FSPF seems to be more predictable. Moreover, we can not confirm that there is a relevant difference between the container's start-up time across scenarios. To gain more confidence on relevance and significance, we needed to use the

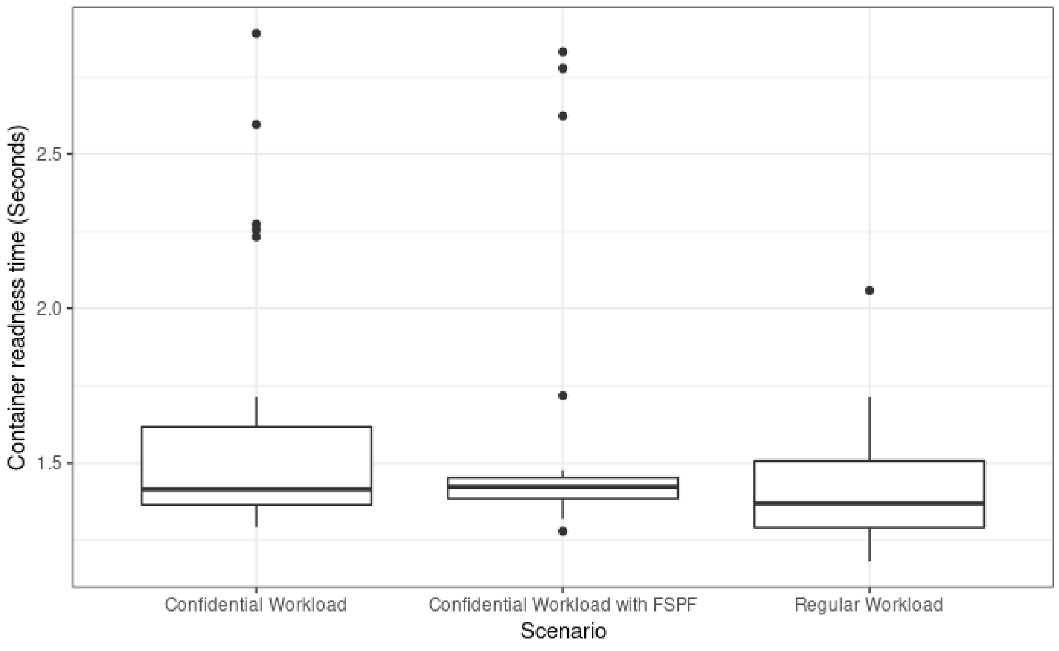


Figure 6.1: Time needed to execute the container entry point (Container readiness).

confidence intervals on the difference between the medians.

The confidence intervals on the difference between the median start-up times are shown in Figure 6.3. Where:

- CW means confidential workload;
- CWwF means Confidential Workload with FSPF;
- RW means Regular Workload.

We can see that the confidence interval on the difference between the median of the start-up times includes the zero for all scenarios. Since the interval contains the zero (no difference), we do not have sufficient evidence to conclude that there is a statistically significant difference.

As illustrated in Figure 6.4, for the confidence intervals on the difference between median SVID delivery times, the confidence intervals do not include the zero for all scenarios. Then, we can say that there is a significant difference between all scenarios. In the median of the SVID delivery time, all the differences considered are relevant. It occurs because, when it comes to confidential workloads, the attestation process is more robust and takes more time.

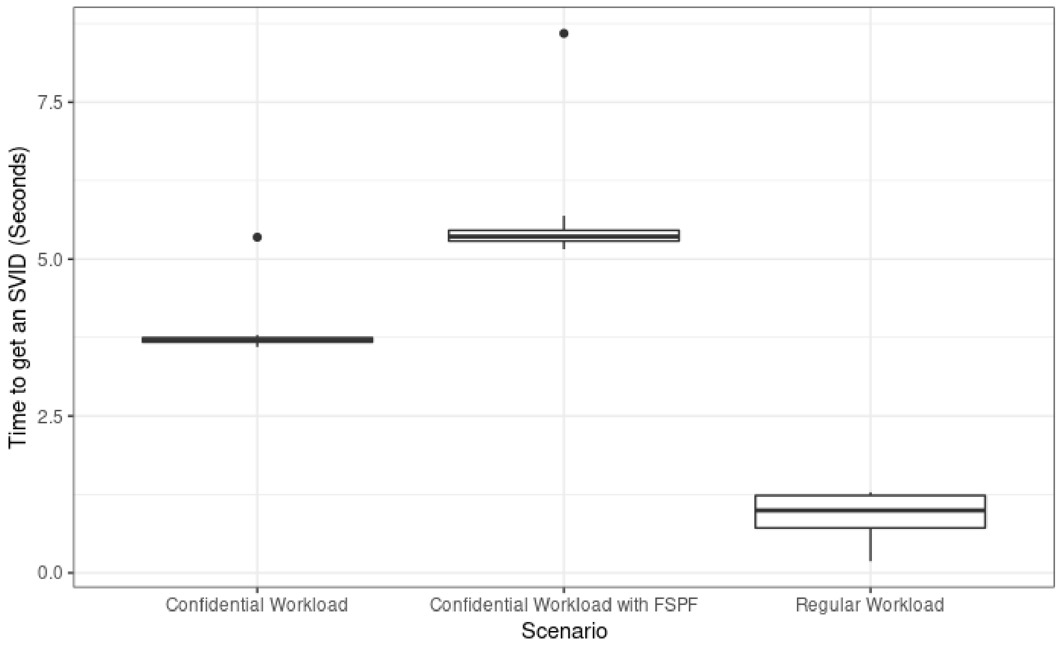


Figure 6.2: Time spent to get an SVID after the container is ready.

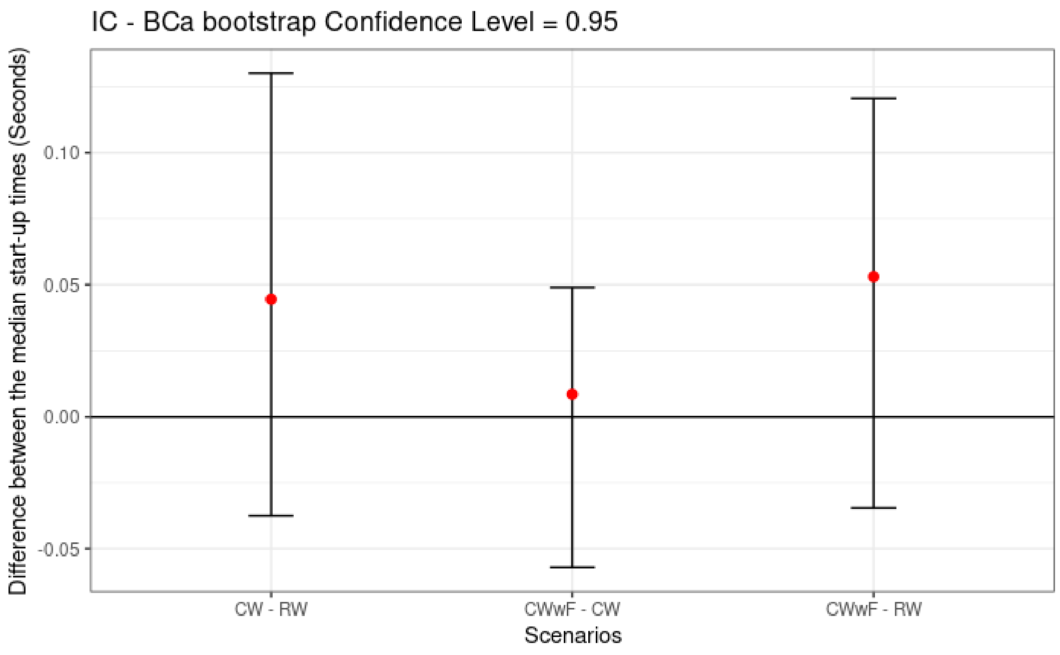


Figure 6.3: Confidence Interval on the difference between the median start-up times.

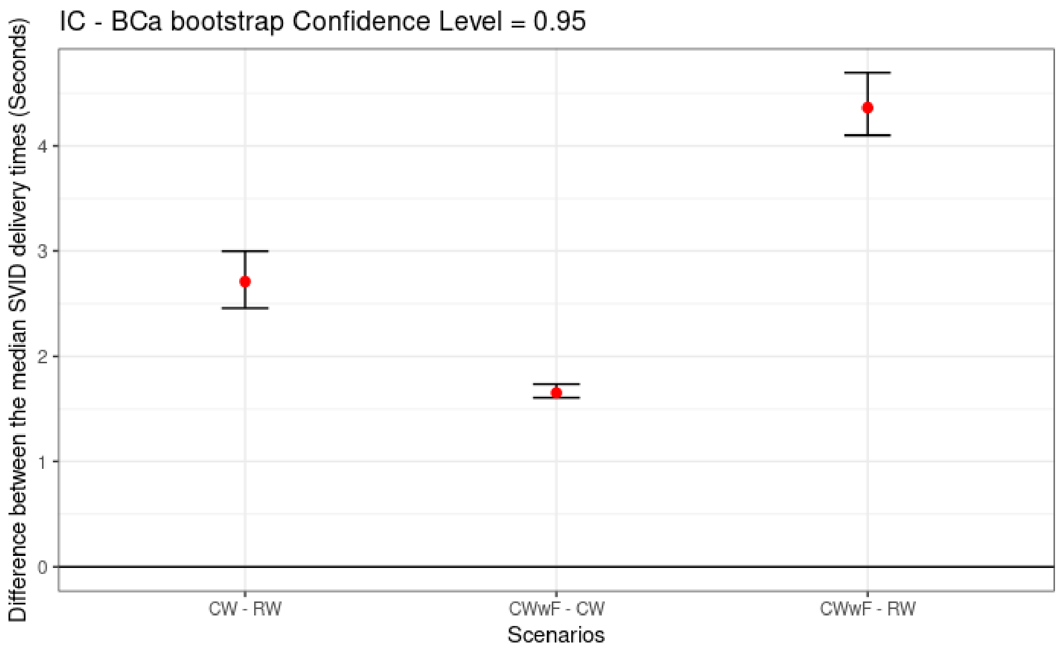


Figure 6.4: Confidence Interval on the difference between the median SVID delivery times.

Also, when using FSPF there is the overhead of the encryption and authentication processes. Still, considering the security constraints ensured by the SCONE attestation, the differences of about 2.7 ($CW - RW$) and 4.3 ($CWwF - RW$) do not make it infeasible for typical container workloads.

In Figure 6.5, we show the total time spent from the execution of the `kubectl` command execution until the workloads gain the SVIDs. This figure clearly shows the impact of the SGX attestation process in the total time. After the workloads receive their SVIDs, the overheads are the same as those studied in previous research, such as the work presented by Krahn et al. [Krahn et al. 2020].

Given the presented results, developers that want to use the approach should be aware that the time needed to attest a confidential workload and give it a new identity is substantial. It can potentially influence the mean time to repair systems with these kinds of workloads as subcomponents. One can apply strategies to enhance fault-tolerance aspects to address this issue and others related to time overhead added by the attestation process. However, this is not the focus of this work.

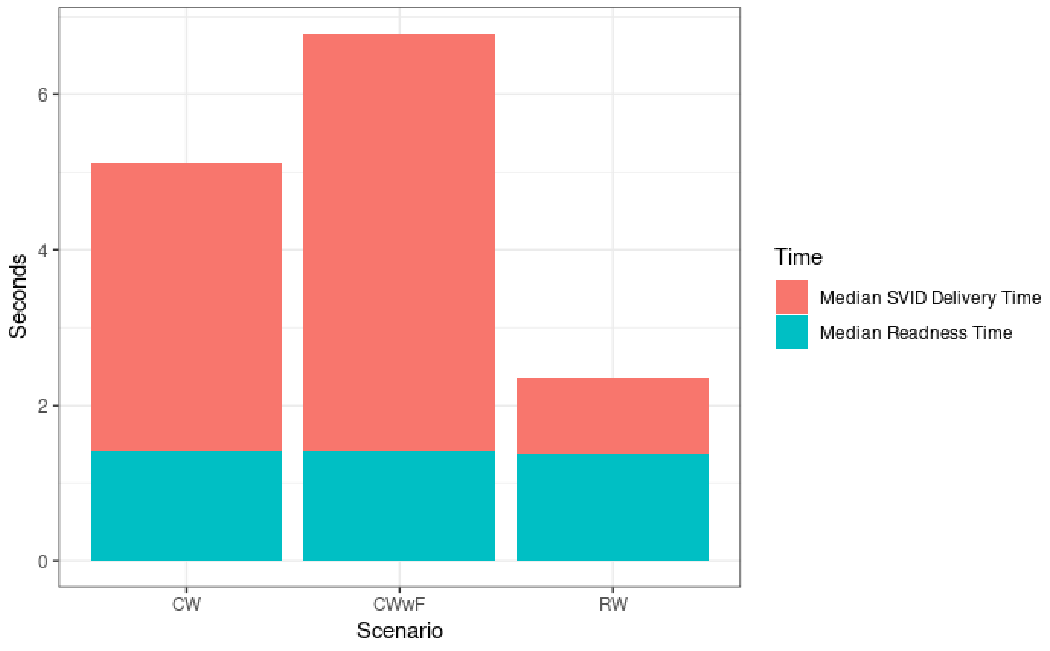


Figure 6.5: Total time spent from the `kubectl apply` command to the time when the workload is with the SVID.

6.2 Image Build Times

Developers have to change the Dockerfiles to use the SCONE approach for turning regular workloads into confidential workloads. In the best case, changing the base image is enough to do the work. In a longer path, the developer must recompile some dependencies with one of the SCONE compilers. Moreover, if the workloads have code confidentiality requirements, the developer may want to use the SCONE's FSPF feature.

Investigating the build time is essential since the enterprise-level CI/CD platforms bills the services per usage minutes. For instance, the CircleCI performance plan starts at 25.000 credits for $15, and it consumes ten credits per minute. Then, we designed an experiment to compare the proposed solution's build times against regular use of SPIRE using Python3 as a reference programming language.

The following research question guided the experiment:

- **RQ3**: Do confidential workload images have a longer build time than regular workloads?

6.2.1 Methodology, Environment, and Tools

To answer the research questions described in the last section, we designed an experiment to compare the proposed solution's build times against SPIRE's regular use using Python3 as a reference programming language. We considered as a factor for this experiment the workload scenario, which will impact directly in the Dockerfiles, and, consequently, in the build time. This factor has three levels:

- Confidential: a Dockerfile for a workload that uses the SGX Helper to get an SVID. This Dockerfile uses the SCONE FSPF feature in the build, which means that, in this scenario, it encrypts the application code and authenticates dependencies at the directories `/lib` and `/usr/lib`.
- NoFSPF: a Dockerfile for a confidential workload written in Python3 uses the SGX Helper to get an SVID. The workload shipped in the Docker image protects only the data, and there are no guarantees about code confidentiality.
- Regular: a Dockerfile for a regular workload written in Python3 uses the SPIRE Agent's workload API to get an SVID.

We executed 30 replicas in a VM with 4 GB of RAM and 2 vCPUs.

6.2.2 Discussion

To enable both a comparison and a good understanding of the times involved, in Figure 6.6, we show the distributions for each scenario's image build times (side by side and isolated). We can see that the times for regular workload builds are much larger than for other scenarios.

The BCa confidence intervals for the image time builds are shown in Figure 6.7. In Figure 6.8, we focus only in the scenarios with confidential workloads either with FSPF or not.

The difference between the build times is significant across all scenarios. Moreover, the regular scenario is significantly larger than the others. The proposed integration between SPIRE and confidential workloads leverages the SCONE CAS to attest confidential workloads giving them SVIDs transparently. It means that the workload call to the Workload API

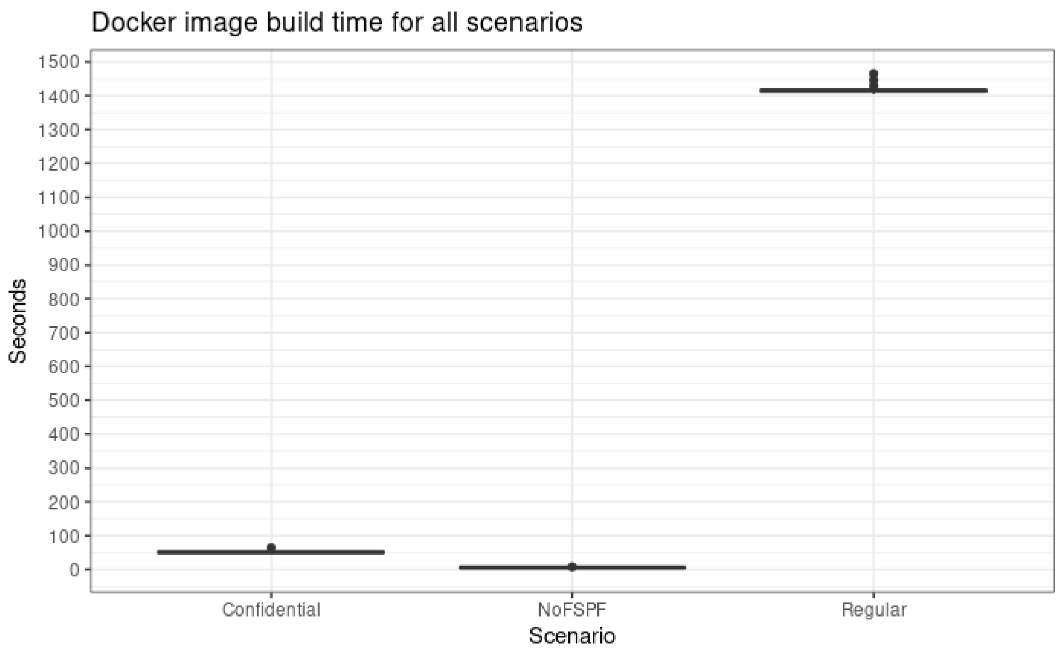


Figure 6.6: Image build time distribution (all scenarios).

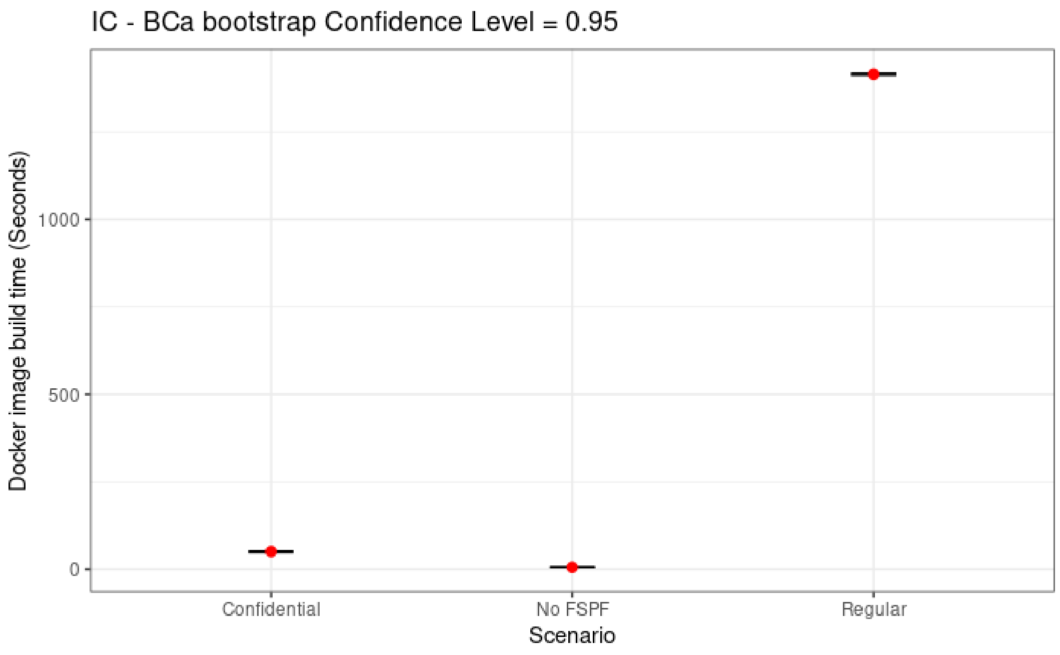


Figure 6.7: Confidence interval on Docker image build time.

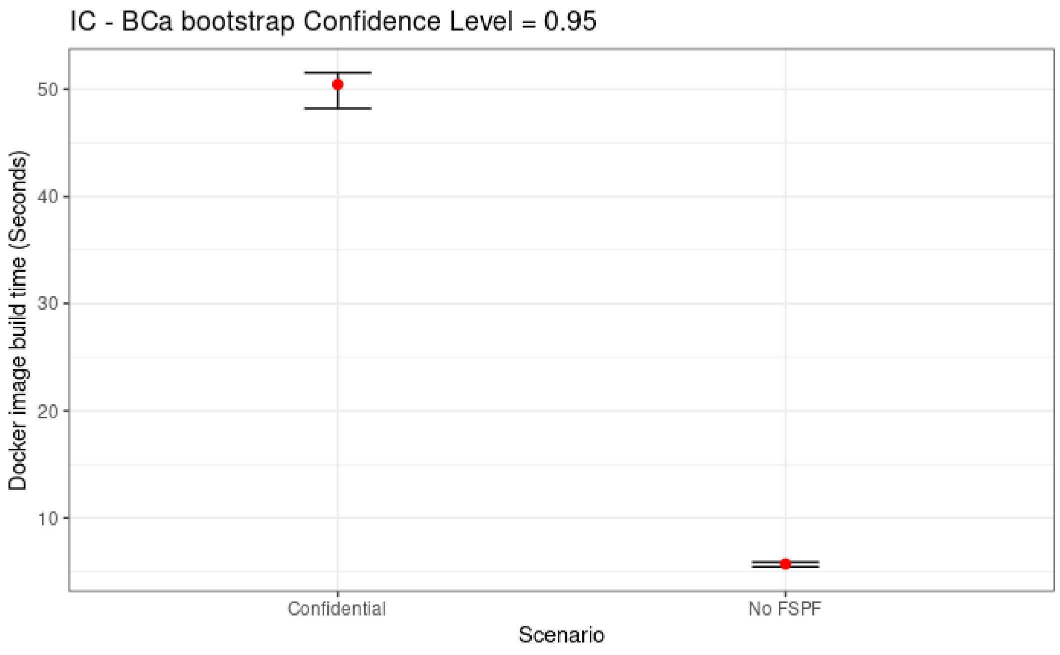


Figure 6.8: Confidence interval on Docker image build time for confidential workloads.

(RPC) is not necessary. The gRPC library and its dependencies have a time-consuming build process, impacting the total build time. It explains the more prominent times in the regular scenario.

Also, in the confidential scenario, the SCONE FSPF encrypts the application code and authenticates the dependencies cryptographically. Because of this, the confidential scenario's build time is slightly longer than the scenario without FSPF. The build time for confidential workloads written in Python3, in this experiment, was less than a half of the build time for regular workloads written in the same programming language, despite the additional security layers added by SCONE.

Chapter 7

Related Work

Previous work also addressed the challenge of provisioning identities to confidential workloads in a way that enables interoperation with regular workloads. This chapter briefly describes these related works and presents a comparison table that summarizes four features of interest: mutual authentication, interoperability, low adoption barrier, and multi-platform attestation. All the approaches described in this chapter consider confidential computing threat model.

7.1 PALÆMON (CAS)

Gregor et al. [Gregor et al. 2020] proposed PALÆMON (another name for CAS) as a trust management service designed to enable trusted workload execution in face of Byzantine stakeholders. PALÆMON can attest workloads deliver secrets (which could be identities) after a successful attestation process. Also, CAS can generate and manage certificates and keys natively.

Besides the capability of attesting workloads, PALÆMON also can convince challengers (operators, users, client applications) that it is running without unintended modifications in an updated SGX environment. However, unlike the SPIFFE framework, the attestation service has an opinionated attestation process that only works with SCONE-enabled workloads, making it unsuitable for regular workloads in cases where software attestation is needed for regular workloads.

7.2 SQUAD

The Secure Simple Storage Service for SGX-based Microservices (SQUAD) enables operators to deliver any secrets to SGX based workloads [da Silva, de Oliveira Silva e Brito 2019]. It is a proof-of-concept that supports only confidential workloads developed with the Intel SGX SDK. It has a pluggable authorization mechanism that can leverage any measurements from the attestation process, such as the MRENCLAVE. Similarly, as with CAS, one can use SQUAD as part of a solution to provide identities for confidential workloads. However, SQUAD does not consider the filesystem and the only way to pass secrets to workloads is via a network connection wrapped by a secure channel established with the attestation process.

7.3 RA-TLS

Combining interoperable certificates and attestation, Knauth et al. proposed integrating a standard TLS communication channel with the Intel SGX remote attestation process [Knauth et al. 2018]. According to Knauth et al., Stumpf et al. [Stumpf et al. 2006] showed that one must integrate the remote attestation and secure communication channel establishment to prevent man-in-the-middle attacks. The SIGMA Protocol [Krawczyk 2003] involved in the attestation process, as described by Anati et al. [Anati et al. 2013], ensures a shared secret is reached at the end of the attestation process. With this shared secret, one can bootstrap a secure channel. However, Knauth claims that it is inefficient since it duplicates work. Then, using the standard TLS secure channels is an alternative.

RA-TLS (Remote Attestation Transport Layer Security) used SGX as a hardware root of trust, including attestation evidence in the existing X.509 certificates. RA-TLS links the TLS key used to bootstrap the channel into the SGX report as user data. In this way, the authors bind the RA-TLS to the attester enclave. The attester put the attestation evidence into the certificate extensions, which requires no changes in existing TLS libraries, but requires the implementation of custom hooks to verify these extensions.

When using RA-TLS, the TLS protocol ensures the freshness of the exchanged messages. Since the enclave builds the attestation evidence and the self-signed certificate in the application startup, a challenger must use additional mechanisms to assess the attestation evidence's freshness. For instance, in the EPID-based (Enhanced Privacy ID) attestation, the attestation verification report has a timestamp that a challenger may use to determine a piece of evidence, signed by the IAS, as too old.

Still considering the freshness of the attestation evidence, in case of a mutual authentication where both entities run inside enclaves, it may not be easy to assess the freshness without a trusted time source. Also, in the RA-TLS approach, the workloads must be aware of the enclave measurements of other workloads it needs to communicate with.

The authors identified some limitations of the proposed solution. Because of non-standard X.509 extensions, clients may abort the connections after finding an unknown critical extension. Extensions should not be marked as critical to avoid this problem and allow backwards compatibility, but legacy clients may still complain about a self-signed certificate. A solution for the self-certificate problem could be having a trusted CA signing the extended certificate (using protocols such as Automated Certificate Management). Also, when using RA-TLS, the certificate size increases significantly (at least six times larger).

7.4 Comparison Table

Table 7.1 summarizes four significant features of the approaches discussed previously. A filled circle denotes a fully supported feature built in the proposed solution. An empty circle indicates that the feature is not supported. A half-filled circle indicates that the feature can be supported with some effort. We consider the following features essential to distinguish the related work:

- **Mutual authentication**: the approach enables two arbitrary workloads to authenticate mutually using some identity artifact, e.g., an X.509 certificate.
- **Low adoption barrier**: this feature considers how easy it is to adopt the approach for its original purpose in terms of workload modification.
- **Multi-platform attestation process**: the approach has an attestation process suitable for both confidential and regular workloads. While the mutual authentication feature concerns using the identity artifact for authentication, this feature concerns the attestation process performed to give such artifact to a workload.

- **Interoperability**: the approach enables mutual authentication between confidential workloads and regular workloads.

Approach \ Features	**Mutual Authn.**	**Interop.**	**Low Adop. Bar.**	**Multi. Attest.**
CAS	◐	●	●	○
SQUAD	◐	◐	○	○
RA-TLS	◐	◐	○	○
SPIFFE + SGX	●	●	●	●

Table 7.1: Related work comparison table.

All the approaches enable operators to set up mutual authentication for workloads through X.509 certificates. However, there are some limitations related to this feature. To support mutual authentication using CAS between a confidential workload and a non-confidential workload, the operator has at least two options: The first is to use an external certificate authority and a strategy to mirror CA certificates into CAS. The second one is to use the CAS as a CA and configure workloads manually or provision the artifacts for mutual authentication through the CAS's sessions API (for instance, the CAs exposed in public sessions). In the case of SQUAD, it cannot manage a PKI itself, even for SGX workloads. An operator using SQUAD must also rely on an external certificate authority. Also, there should be a component to store artifacts for mutual authentication into SQUAD. The RA-TLS relies on non-standard X.509 extensions, which can make non-confidential workloads abort the connections. To avoid the aborted connections, the extensions could be marked as not critical ones. The SPIFFE-SGX integration leverages the SPIRE implementation to provide all the artifacts needed to enable mutual authentication. Moreover, a mutual authentication process between a confidential and a non-confidential workload occurs seamlessly on both sides using SPIFEE-SGX.

The interoperability feature is closely linked to the mutual attestation one. Offering built-in support for confidential workloads to identify non-confidential workloads and vice versa is a challenging problem. Using CAS, one can configure non-confidential workloads with certificates to trust CAS as a certificate authority. Following this path, there is no attestation process for non-confidential workloads. When it comes to using SQUAD, one can not use it as a CA with built-in capabilities. However, it is possible to implement such functionalities throughout the SQUAD plugins workflow. Likewise, because of the limitations discussed previously, the RA-TLS approach can not offer interoperability by itself. It needs some additional efforts to circumvent limitations, mainly to handle the extensions and include the certificate generated by the enclaves into a certificate chain. In contrast with the related works, the SPIFFE-SGX integration enables confidential workloads to identify non-confidential and vice versa. Furthermore, our proposal inherits the attestation process of the SPIFFE implementation. This process does not ensure the same trust level as the SGX attestation process, but it is still a desirable security layer.

Regarding the adoption barrier, adopting CAS is straightforward. Once the operator puts a workload into an enclave, using the SCONE lift-and-shift approach, the workload does not need to be changed. The SCONE environment passes the secrets to applications via environment variables, command-line arguments, and files, which are the most popular ways to configure an application as verified by Gregor et al. [Gregor et al. 2020]. On the other hand, the adoption barrier for SQUAD is high. To use the SQUAD to deliver secrets for SGX-SDK applications, one has to implement a custom secrets delivery function that uses the secure communication channel resultant from the attestation process (or at least a library to link against the enclave that extends the attestation process library). For a lift-and-shift approach of running workloads inside enclaves, one would have to implement additional attestation procedures and plugins. Similarly, to adopt the RA-TLS approach, one must extend the TLS libraries used by the workloads to recognize the new extensions and verify the attestation evidence within these extensions.

To analyze the adoption barrier for our proposed approach, we have to split the adoption barrier into two fronts: SPIFFE's adoption barrier and CAS's adoption barrier.

Suppose an operator wants to adopt SPIRE, the SPIFFE implementation. In that case, some work is needed to make the applications call the Workload API, so these applications get attested and receive an SVID. An approach to avoid these problems and keep the workloads unaware of SPIFFE is to use sidecar proxies such as Envoy [1]. In this way, the workload does not concern TLS connections, authorization, and authentication. In general, sidecars are next to the workloads within an orchestrator's basic deployable objects (Pods, in the case of

[1]https://www.envoyproxy.io/

Kubernetes).

Gregor et al. [Gregor et al. 2020] explain that, because of the CAS design goals, the CAS (and the SCONE environment) gives all sorts of secrets via existing common ways to provide configurations to workloads. So, in the lucky path, the operator needs to change only deployment files, and container image build files. In the more challenging course, the operator may need to find alternatives to dependencies not supported by the TEE environment.

With the integration between SPIFFE and SGX, supported by the SCONE lift-and-shift approach, we reduce SPIRE's adoption barrier by leveraging CAS to deliver the identities to workloads. In other words, it is possible to receive SVIDs via protected filesystem file injection. CAS will condition the SVID injection in the workload's view of the filesystem to the robust SGX attestation process.

Finally, the SPIFFE-SGX integration is the only approach that supports multi-platform attestation thanks to the pluggable architecture of the SPIFFE implementation (SPIRE). The other approaches have attestation processes and do not support attestation of non-confidential workloads. In other words, these other approaches do not support attesting workloads with a lower level of trust.

Chapter 8

Conclusion

In this chapter, we present, in Section 8.1, a summary of our work. Besides, Section 8.2 introduces future work.

8.1 Summary

In this work, we proposed an approach to integrate the SCONE environment and the SPIFFE framework. To do so, we added a new component to SPIRE, the so-called SGX Helper, enabling universal identity support for confidential workloads.

We conducted a security analysis to understand the effects of the threat model for confidential computing workloads on the current SPIRE implementation. The analysis revealed that the current implementation does not fit this more aggressive threat model. It also revealed that our proposal mitigates most of the attacks coming from untrusted infrastructure providers and identity theft attacks coming from workloads.

The experiments performed showed an overhead to deliver identities to workloads, which is the price to pay for the robust SGX attestation process. On the other hand, the image build time for the Python reference workloads used in the experiments is shorter for the confidential workloads. It occurred because of the time-consuming gRPC builds in regular workloads needed to consume the Workload API. Among the confidential workload builds, the builds with SCONE FSPF, required for protecting code, pre-loaded data and libraries, presented significant time overhead.

We discussed the proposal with the community[1], ensuring its applicability in real-world scenarios and its compatibility with the SPIFFE standards. In this sense, the contributions of this work were essential to the current in-progress effort to give SPIRE the support for confidential workloads, impacting future releases of both SPIRE and SCONE.

8.2 Future Work

During the security analysis, it became evident that the Agent in the same node, along with the Server, weakens the Agent-Server security boundary. In recent work, we put the Agent and the Server inside SGX enclaves. As enclaves are isolated from others, the security boundary is hardened by the TEE. With the Agent and Server running inside SGX enclaves, it is necessary to perform a new security analysis to assert the benefits and understand the big picture. Moreover, the SGX security benefits, in general, come with a performance overhead. Thus, we should conduct new performance experiments.

The difficulty to access the Workload API for some types of workloads is not specific to the SCONE workloads. In fact, for serverless workloads (and also envisaging other similar cases), the community has adopted a similar approach. The community designed a new type of plugin, called SVIDStore, to handle SVIDs and push them into trusted third-party components. Then, the third-party storages will act as intermediates, storing the SVIDs received from the agent and handling them to the special workloads using case-specific protocols or approaches. The first plugin implementations will cover Azure, AWS, and Google Cloud.

The new SVIDStore plugin type can accommodate the SGX Helper logic, with a smaller codebase and less implementation effort. Currently, we have a work in progress to enable identity issuing for confidential workloads, implementing the results of this dissertation using an SVIDStore plugin.

[1]Example of an issue discussed with the community: https://github.com/spiffe/spire/issues/1924.

Bibliography

[Anati et al. 2013]ANATI, I. et al. Innovative technology for cpu based attestation and sealing. In: ACM NEW YORK, NY, USA. *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*. [S.l.], 2013. v. 13, p. 7.

[Arnautov et al. 2016]ARNAUTOV, S. et al. SCONE: Secure linux containers with intel SGX. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016. p. 689–703. ISBN 978-1-931971-33-1. Disponível em: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov>.

[Costan e Devadas 2016]COSTAN, V.; DEVADAS, S. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, v. 2016, n. 86, p. 1–118, 2016.

[da Silva, de Oliveira Silva e Brito 2019]da Silva, M. S. L.; de Oliveira Silva, F. F.; Brito, A. Squad: A secure, simple storage service for sgx-based microservices. In: *2019 9th Latin-American Symposium on Dependable Computing (LADC)*. [S.l.: s.n.], 2019. p. 1–9.

[Demigha e Larguet 2021]DEMIGHA, O.; LARGUET, R. Hardware-based solutions for trusted cloud computing. *Computers & Security*, v. 103, p. 102117, 2021. ISSN 0167-4048. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167404820303904>.

[Feldman et al. 2020]FELDMAN, D. et al. *Solving the Bottom Turtle — a SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity*. 1. ed. [S.l.: s.n.], 2020. This book presents the SPIFFE standard for service identity, and SPIRE, the reference implementation for SPIFFE. https://spiffe.io/book/.

[Gannon, Barga e Sundaresan 2017]Gannon, D.; Barga, R.; Sundaresan, N. Cloud-native applications. *IEEE Cloud Computing*, v. 4, n. 5, p. 16–21, 2017.

[Greenwood 2014]GREENWOOD, M. C. Intermediate statistics with r. In: ____. 2. ed. [S.l.: s.n.], 2014. p. 74.

[Gregor et al. 2020]Gregor, F. et al. Trust management as a service: Enabling trusted execution in the face of byzantine stakeholders. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. [S.l.: s.n.], 2020. p. 502–514.

[IBM Security. Cost of a Data Breach Report 2020. https://www.ibm.com/security/data-breach] IBM Security. Cost of a Data Breach Report 2020. https://www.ibm.com/security/data-breach. Disponível em: <https://www.ibm.com/security/data-breach>.

[Knauth et al. 2018]KNAUTH, T. et al. Integrating remote attestation with transport layer security. *CoRR*, abs/1801.05863, 2018. Disponível em: <http://arxiv.org/abs/1801.05863>.

[Krahn et al. 2020]KRAHN, R. et al. Teemon: A continuous performance monitoring framework for tees. In: *Proceedings of the 21st International Middleware Conference*. New York, NY, USA: Association for Computing Machinery, 2020. (Middleware '20), p. 178–192. ISBN 9781450381536. Disponível em: <https://doi.org/10.1145/3423211.3425677>.

[Krawczyk 2003]KRAWCZYK, H. Sigma: The 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike protocols. In: BONEH, D. (Ed.). *Advances in Cryptology - CRYPTO 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. p. 400–425. ISBN 978-3-540-45146-4.

[Microsoft Azure: Confidential Computing Documentation.]MICROSOFT Azure: Confidential Computing Documentation. Disponível em: <https://docs.microsoft.com/en-us/azure/confidential-computing/>.

[Oleksenko et al. 2018]OLEKSENKO, O. et al. Varys: Protecting {SGX} enclaves from practical side-channel attacks. In: *2018 {Usenix} Annual Technical Conference ({USENIX}{ATC} 18)*. [S.l.: s.n.], 2018. p. 227–240.

[Sabt, Achemlal e Bouabdallah 2015]Sabt, M.; Achemlal, M.; Bouabdallah, A. Trusted execution environment: What it is, and what it is not. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. [S.l.: s.n.], 2015. v. 1, p. 57–64.

[Scarlata et al. 2018]SCARLATA, V. et al. Supporting third party attestation for intel® sgx with intel® data center attestation primitives. *White paper*, Intel Corporation, 2018.

[Schwarz, Weiser e Gruss 2019]SCHWARZ, M.; WEISER, S.; GRUSS, D. Practical enclave malware with intel sgx. In: SPRINGER. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. [S.l.], 2019. p. 177–196.

[SPIFFE: Secure Production Identity Framework for Everyone. https://spiffe.io]SPIFFE: Secure Production Identity Framework for Everyone. https://spiffe.io. Disponível em: <https://spiffe.io/>.

[Stumpf et al. 2006]STUMPF, F. et al. A robust integrity reporting protocol for remote attestation. In: CITESEER. *Second Workshop on Advances in Trusted Computing (WATC'06 Fall)*. [S.l.], 2006. p. 25–36.