UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA – PPGCAP

BEATRIZ MICHELSON REICHERT

IT-SPIRE: UM SERVIDOR SPIRE TOLERANTE A INTRUSÕES

JOINVILLE
2024

BEATRIZ MICHELSON REICHERT

IT-SPIRE: UM SERVIDOR SPIRE TOLERANTE A INTRUSÕES

Dissertação apresentada ao Programa de Pós–Graduação em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada.

Orientador: Dr. Rafael Rodrigues Obelheiro

JOINVILLE
2024

Ficha catalográfica elaborada pelo programa de geração automática da

Biblioteca Universitária Udesc,

com os dados fornecidos pelo(a) autor(a)

Reichert, Beatriz Michelson
IT-SPIRE: um servidor SPIRE tolerante a intrusões / Beatriz Michelson Reichert. -- 2024.
59 p.

Orientador: Rafael Rodrigues Obelheiro
Dissertação (mestrado) -- Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada, Joinville, 2024.

1. SPIFFE/SPIRE. 2. Tolerância a intrusões. 3. Gerenciamento de identidades. I. Obelheiro, Rafael Rodrigues. II. Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Programa de Pós-Graduação em Computação Aplicada. III. Título.

BEATRIZ MICHELSON REICHERT

IT-SPIRE: UM SERVIDOR SPIRE TOLERANTE A INTRUSÕES

Dissertação apresentada ao Programa de Pós–Graduação em Computação Aplicada do Centro de Ciências Tecnológicas da Universidade do Estado de Santa Catarina, como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada.

Orientador: Dr. Rafael Rodrigues Obelheiro

BANCA EXAMINADORA:

Dr. Rafael Rodrigues Obelheiro
Universidade do Estado de Santa Catarina (UDESC)

Membros:

Dr. Maurício Aronne Pillon
Universidade do Estado de Santa Catarina (UDESC)

Dr. Eduardo Adilio Pelinson Alchieri
Universidade de Brasília (UnB)

Joinville, 22 de agosto de 2024

Dedico este trabalho à minha família, amigos e professores que me apoiaram nessa trajetória.

AGRADECIMENTOS

Agradeço à minha família pelo apoio, paciência e motivação, sem estes eu não chegaria até aqui. Ao meu namorado pela paciência, e por compreender a necessidade de usar os finais de semana para estudar. Aos meus companheiros de quatro patas, por me proporcionarem momentos de distração e diversão. Aos meus amigos por não me deixarem desanimar durante toda essa caminhada.

Ao meu orientador, o professor Dr. Rafael Rodrigues Obelheiro, pela orientação, e disponibilidade, mesmo no período de férias. Destaco a paciência e o empenho dedicado à elaboração deste trabalho. Serei eternamente grata pelo apoio.

Agradeço também ao professor Dr. Maurício Aronne Pillon pelo suporte no LabP2D, até mesmo fora do horário de trabalho. Ao Eduardo Alchieri e ao Caio Costa pelo auxílio com o BFT-SMaRt. Ao Agustín Martínez Fayó e Andrew Harding por esclarecerem as minhas dúvidas no grupo do SPIFFE no Slack.

À Fundação de Amparo à Pesquisa e Inovação do Estado Santa Catarina (FAPESC) e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Código de Financiamento 001 pelo financiamento deste trabalho.

Por fim, a todos que de algum modo fizeram parte desta caminhada, o meu muito obrigada!

“Don’t Panic.” (Douglas Adams)

RESUMO

Uma identidade de software é um conjunto de atributos que identificam componentes de software em um ambiente de aplicações distribuídas, e que pode ser usada para autenticação e autorização. O SPIRE é a implementação de referência do SPIFFE, um arcabouço para gerenciamento de identidades de software em ambientes dinâmicos e heterogêneos. A arquitetura do SPIRE contém alguns componentes críticos que possuem confiança total e que podem afetar severamente a segurança de um sistema caso sua disponibilidade ou integridade seja comprometida. Para melhorar a resiliência da arquitetura SPIRE, esta dissertação introduz o IT-SPIRE, um servidor SPIRE tolerante a intrusões baseado em replicação máquina de estados tolerante a falhas bizantinas (TFB). Duas características importantes do IT-SPIRE são *(i)* o uso de *proxies* que encapsulam protocolos TFB, permitindo minimizar alterações no código do SPIRE, e *(ii)* permitir que os clientes usem *tokens*/certificados de identidade não modificados (SVIDs). Por conseguinte, foi realizada a implementação de um protótipo do IT-SPIRE para demonstrar sua viabilidade e obter uma estimativa preliminar de seu impacto em termos de desempenho. Observou-se que o protótipo do IT-SPIRE possui um tempo de inicialização maior que o SPIRE, com um tempo de sincronização comparável. Concluiu-se que o IT-SPIRE tem custos de desempenho razoáveis que são compatíveis com a sua segurança reforçada: enquanto o SPIRE pode ser usado em uma arquitetura de implantação capaz de tolerar faltas de parada do servidor, o IT-SPIRE tolera faltas e intrusões tanto no servidor quanto no banco de dados.

Palavras-chave: SPIFFE/SPIRE. Tolerância a intrusões. Gerenciamento de identidades.

ABSTRACT

A software identity is a set of attributes that identify software components in a distributed application environment, and which may be used in authentication and authorization. SPIRE is the reference implementation of SPIFFE, a framework for managing software identities in dynamic and heterogeneous environments. The SPIRE architecture contains some critical components that are fully trusted and can severely affect the security of a system if their availability or integrity is compromised. To improve the resilience of the SPIRE architecture, in this dissertation, we introduce IT-SPIRE, an intrusion-tolerant SPIRE Server based on Byzantine fault-tolerant (BFT) state machine replication. Two important characteristics of IT-SPIRE are *(i)* its use of proxies that encapsulate BFT protocols, which allows us to minimize changes to the SPIRE code, and *(ii)* allowing clients to use unmodified identity tokens/certificates (SVIDs). We implemented a prototype of IT-SPIRE to demonstrate its feasibility and gauge its performance impact. We observed that IT-SPIRE has a longer initialization time than SPIRE, with a comparable synchronization time. We conclude that IT-SPIRE has reasonable performance costs that are compatible with its enhanced security: while SPIRE may be used with a deployed architecture capable of tolerating server crash faults, IT-SPIRE tolerates faults and intrusions in the server and/or in the database.

Keywords: SPIFFE/SPIRE. Intrusion tolerance. Identity management.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AC	Autoridade Certificadora
API	*Application Programming Interface*
BD	Banco de Dados
BFT	*Byzantine Fault Tolerance*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CLI	*Command-line Interface*
CSR	*Certificate Signing Request*
CV	Coeficiente de Variação
DNS	*Domain Name System*
DoS	*Denial Of Service*
DVID	*Delegated Assertion SVID*
FAPESC	Fundação de Amparo à Pesquisa e Inovação do Estado Santa Catarina
FFI	*Foreign Function Interface*
IC	Intervalo de Confiança
IdP	*Identity Provider*
IT	*Intrusion Tolerance*
JWT	*JavaScript Web Token*
MITM	*Man-in-the-Middle*
mTLS	*Mutual Transport Layer Security*
PBFT	*Practical Byzantine Fault Tolerance*
PVSS	*Publicly Verifiable Secret Sharing*
RME	Replicação Máquina de Estados
RPC	*Remote Procedure Call*
RSA	*Rivest-Shamir-Adleman*
SCADA	*Supervisory Control and Data Acquisition*
SGX	*Software Guard Extensions*
SMR	*State Machine Replication*
SP	*Service Provider*
SPIFFE	*Secure Production Identity Framework for Everyone*
SPIRE	*SPIFFE Runtime Environment*

SSH	*Secure Shell*
SSO	*Single Sign-on*
STRIDE	*Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege*
SVID	*SPIFFE Verifiable Identity Document*
TF	Tolerância a Faltas
TFB	Tolerância a Falhas Bizantinas
TI	Tolerância a Intrusões
UDESC	Universidade do Estado de Santa Catarina
UnB	Universidade de Brasília
URI	*Uniform Resource Identifier*
VSS	*Verifiable Secret Sharing*
ZTA	*Zero–Trust Architecture*

SUMÁRIO

1 INTRODUÇÃO

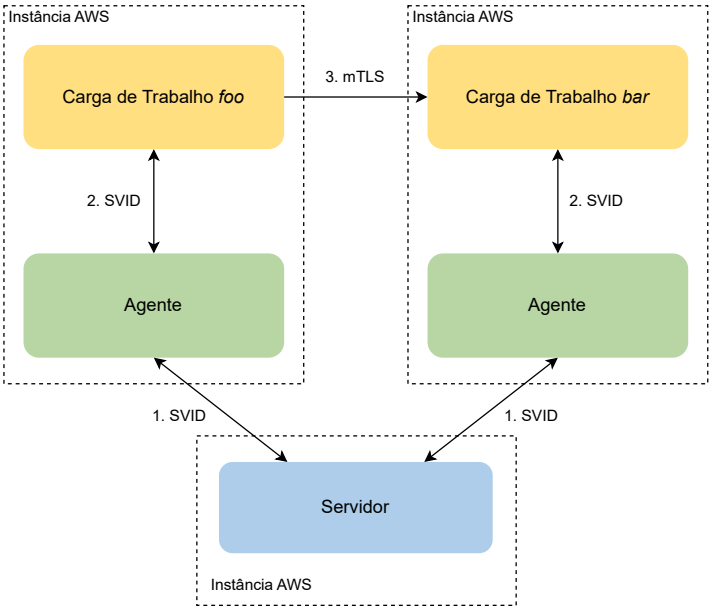
A arquitetura de confiança zero (*Zero-Trust Architecture*, ZTA) (ROSE et al., 2020) vem ganhando adoção para garantir a segurança de aplicações distribuídas, particularmente devido ao uso crescente de nuvens computacionais. A ZTA propõe a eliminação da noção de perímetros de segurança em que os componentes de software confiam uns nos outros, e seus preceitos incluem a autenticação mútua de componentes e o uso pervasivo de canais criptografados de comunicação, o que exige mecanismos seguros para emissão e validação de identidades de software. Uma identidade de software consiste em um conjunto de atributos exclusivos, os quais são usados para autenticar o software e autorizar seu acesso a determinados recursos (MICROSOFT, 2024).

Suponhamos que um serviço *foo* deseje se comunicar com um serviço *bar*, ambos localizados em instâncias AWS distintas. Aqui, observa-se uma ameaça de falsificação de servidor (REICHERT; OBELHEIRO, 2022), *i.e.*, a possibilidade de que o serviço *foo* seja redirecionado para um serviço *bar* ilegítimo que assume o lugar do verdadeiro. A arquitetura ZTA visa mitigar essa ameaça, e um método usado para autenticação mútua entre processos é o mTLS (*Mutual Transport Layer Security*). Entretanto, este possui alguns desafios como: *bootstrapping* de confiança, provisionamento, revogação, rotação e monitoramento de chaves/certificados para serviços (SIRIWARDENA; DIAS, 2020). Para se autenticar usando mTLS, é necessário que um serviço tenha um par de chaves (pública e privada) e apresente um certificado assinado por uma autoridade certificadora (AC) na qual todos os serviços confiem. Além disso, é necessário garantir a segurança da AC para impedir a emissão de certificados forjados que serão aceitos como legítimos, oportunizando ataques de MITM (*man-in-the-middle*) e falsificação de servidor (REICHERT; OBELHEIRO, 2022). Ademais, há a rotação de certificados, a qual acarreta na geração de um novo certificado para cada serviço dentro de um intervalo de tempo predefinido. Em resumo, uma quantidade considerável de certificados são usados na implementação do mTLS e o gerenciamento desses certificados se torna um desafio para os implementadores. O SPIFFE (*Secure Production Identity Framework for Everyone*), um arcabouço de gerenciamento de identidades de software em ambientes dinâmicos e heterogêneos (FELDMAN et al., 2020), visa resolver os problemas de provisionamento e rotação de chaves e de *bootstrapping* de confiança de forma mais simples. O arcabouço dispensa o uso de segredos compartilhados na autenticação, usando mecanismos de atestação para estabelecer automaticamente a identidade de componentes. O SPIFFE possui uma implementação de referência chamada SPIRE (*SPIFFE Runtime Environment*) (SPIFFE, 2019c).

Considerando o cenário apresentado anteriormente, um serviço, chamado no SPIFFE de Carga de Trabalho, interage com um Agente para solicitar sua credencial. Esse Agente executa localmente e realiza a atestação da Carga por meio de informações do seu sistema operacional e da sua plataforma de nuvem; assim, o Agente é capaz de autenticar suas Cargas de Trabalho sem que estas precisem ser provisionadas com quaisquer credenciais, como segredos compartilhados. Se a atestação for bem sucedida, o Agente retorna a devida credencial para

a Carga. Na arquitetura SPIFFE/SPIRE, a credencial usada para provar a identidade de uma Carga é chamada de SVID (*SPIFFE Verifiable Identity Document*). O Agente é responsável por solicitar ao Servidor os SVIDs das Cargas para as quais este está autorizado, bem como fornecer SVIDs às suas Cargas quando demandado. Dessa forma, após as Cargas *foo* e *bar* recuperarem suas credenciais, estas podem se autenticar e estabelecer conexão mTLS entre si. Já o Servidor SPIRE é responsável por realizar a atestação dos Agentes, e emitir credenciais para as Cargas de Trabalho *foo* e *bar*. A Figura 1 apresenta a arquitetura do cenário relatado usando SPIFFE/SPIRE.

Figura 1 – Cenário de ZTA usando SPIFFE/SPIRE



Fonte: Elaborado pela autora (2024).

Entretanto, a arquitetura do SPIFFE/SPIRE conta com alguns componentes críticos, que são considerados confiáveis; o comprometimento da integridade ou disponibilidade desses componentes pode afetar severamente a segurança de uma aplicação que usa o SPIRE. Um exemplo é o Servidor: caso ele fique indisponível, não será possível emitir novos SVIDs, e os Agentes não poderão atender às suas Cargas. Já o comprometimento da integridade do Servidor pode levar à geração de SVIDs arbitrários, permitindo a atestação de Cargas maliciosas.

Uma forma de eliminar ou diminuir a dependência de componentes confiáveis em um sistema é a tolerância a intrusões, uma abordagem que visa garantir o funcionamento correto de um sistema distribuído mesmo que algumas partes desse sistema sejam comprometidas (VERÍSSIMO; NEVES; CORREIA, 2003). O primeiro passo para o desenvolvimento de sistemas tolerantes a intrusões é eliminar pontos únicos de falhas, *i.e.*, componentes que podem comprometer o sistema como um todo ao ter a sua segurança violada (DESWARTE; POWELL, 2006). Um bloco de construção muito usado na implementação de sistemas tolerantes a intrusões é a

Replicação Máquina de Estados (RME) (SCHNEIDER; ZHOU, 2005). Esta consiste em replicar componentes que implementam serviços para a aplicação, de modo que esta possa continuar funcionando corretamente mesmo diante da falha de alguns desses componentes.

O objetivo desta dissertação de mestrado é propor uma arquitetura tolerante a intrusões com vistas a reduzir os componentes críticos na arquitetura do SPIRE. As contribuições da dissertação podem ser definidas como:

1. A identificação dos componentes críticos do SPIRE, descrevendo como o seu comprometimento pode impactar a integridade e a disponibilidade de aplicações;
2. A proposição do IT-SPIRE, um Servidor SPIRE tolerante a intrusões, e a sua integração com a arquitetura atual do SPIRE; e
3. A implementação de um protótipo do IT-SPIRE e sua avaliação experimental.

1.1 OBJETIVOS

Este trabalho tem como objetivo geral propor o IT-SPIRE, um Servidor SPIRE tolerante a intrusões. Esse objetivo geral desdobra-se nos seguintes objetivos específicos:

1. Identificar os componentes críticos do SPIRE;
2. Projetar uma arquitetura de Servidor SPIRE TI;
3. Implementar um protótipo da arquitetura proposta, de modo a demonstrar sua exequibilidade; e
4. Avaliar experimentalmente o protótipo, visando obter uma quantificação inicial do impacto da arquitetura proposta sobre o desempenho do SPIRE.

1.2 METODOLOGIA

Os métodos de pesquisa adotados neste trabalho são pesquisa referenciada e pesquisa aplicada. Inicialmente foi realizada uma revisão sobre a área de tolerância a intrusões, na qual foram discutidas arquiteturas que fazem uso de diversos blocos de construção, e diferentes contextos de aplicação. Em seguida foi realizado um levantamento dos principais conceitos do SPIFFE, e da arquitetura do SPIRE. Em paralelo, foi realizada uma revisão bibliográfica com o intuito de encontrar trabalhos com temas semelhantes que pudessem auxiliar no desenvolvimento do presente trabalho. Aplicando todos os conceitos aprendidos nas etapas anteriores, identificou-se os componentes críticos do SPIRE e como estes impactam a segurança de aplicações. O próximo passo foi projetar e implementar o IT-SPIRE, um Servidor SPIRE tolerante a intrusões. Por fim, o Servidor IT-SPIRE foi implantado na rede da UDESC para realizar a avaliação experimental do protótipo.

1.3 ORGANIZAÇÃO DO TEXTO

Este documento está organizado em cinco capítulos. O Capítulo 2 apresenta uma revisão de literatura que abrange a fundamentação teórica necessária para o entendimento do trabalho e a discussão de trabalhos relacionados. O Capítulo 3 efetua uma análise dos componentes críticos na arquitetura do SPIRE e detalha a proposta do servidor SPIRE tolerante a intrusões. O Capítulo 4 descreve a implementação e a avaliação experimental do protótipo. Por fim, o Capítulo 5 resume as realizações deste trabalho e apresenta as conclusões obtidas com o desenvolvimento do mesmo.

2 REVISÃO DE LITERATURA

Este capítulo revisa duas questões principais relacionadas a este trabalho. A Seção 2.1 apresenta a área de tolerância a intrusões, discutindo arquiteturas e diferentes contextos de aplicação. A Seção 2.2 discorre sobre Replicação Máquina de Estados, uma abordagem utilizada na proposta desta dissertação. A Seção 2.3 revisa os conceitos da tecnologia SPIFFE/SPIRE. A Seção 2.4 discute os trabalhos relacionados. Por fim, a Seção 2.5 apresenta algumas considerações sobre o que foi explanado nesse capítulo.

2.1 TOLERÂNCIA A INTRUSÕES

A tolerância a intrusões (TI) consiste em aplicar o paradigma da tolerância a faltas (TF) no domínio da segurança (CORREIA, 2005). A ideia é organizar e gerenciar um sistema de modo que uma intrusão em uma parte de um sistema não afete sua segurança como um todo (DESWARTE; POWELL, 2006). Em segurança, é comum haver em um sistema componentes que são presumidos confiáveis; um exemplo seria uma Autoridade Certificadora (AC), que emite certificados que atestam a identidade de usuários ou processos do sistema. Em geral, pressupõe-se que uma AC seja confiável, ou seja, que sua chave de assinatura fique protegida, e que ela emita apenas certificados genuínos. Caso um atacante tenha sucesso em comprometer uma AC, este pode emitir certificados fajutos, que, por terem assinaturas válidas da AC, serão aceitos como legítimos pelos integrantes do sistema. Em outras palavras, se a segurança do sistema depende da segurança dos certificados, o comprometimento da AC compromete a segurança do sistema como um todo. Por isso, geralmente são envidados esforços para tornar a autoridade certificadora à prova de ataques.

Uma autoridade certificadora tolerante a intrusões, por sua vez, seria construída de modo diferente. A funcionalidade da AC seria particionada entre várias réplicas, de modo que a emissão de um certificado válido exigiria a cooperação de pelo menos um subconjunto dessas réplicas. Assim, a AC manteria o seu funcionamento correto mesmo que uma minoria de réplicas fossem comprometidas. A TI não dispensa a proteção dos componentes de um sistema mas o torna mais robusto, a ponto de sobreviver a violações dos mecanismos de proteção.

A tolerância a intrusões é melhor compreendida resgatando-se alguns conceitos de tolerância a faltas, como a distinção entre falha, erro e falta (AVIŽIENIS et al., 2014):

- Falha (*failure*): um evento que ocorre quando um sistema se comporta de modo a desviar da sua especificação de serviço, ou seja, o sistema deixa de fornecer o serviço ou o fornece de modo incorreto.

- Erro: a parte do estado do sistema que pode levar a uma falha; uma falha ocorre quando um erro afeta o serviço do sistema.

- Falta (*fault*): a causa verificada ou presumida de um erro. Uma falta é ativa quando produz um erro, e passiva caso contrário.

Existe uma relação causal entre faltas, erros e falhas:

falta $\rightarrow$ erro $\rightarrow$ falha $\rightarrow$ falta $\rightarrow$ erro $\rightarrow$ falha $\rightarrow \ldots$

Um erro é provocado pela ativação de uma falta; esta falta pode ser uma falta interna que estava inativa e foi ativada por alguma condição, ou pode ser uma falta externa. Este erro pode ser propagado internamente (dentro dos limites de um componente do sistema) ou externamente (quando atinge outros componentes através da interface do serviço). Quando um erro atinge a interface do serviço e o altera de forma inaceitável, tem-se a ocorrência de uma falha. A falha de um componente causa uma falta transiente ou permanente no sistema que contém este componente, e a falha de um sistema causa uma falta externa (transiente ou permanente) nos sistemas que com ele interagem.

Um sistema tolerante a faltas, portanto, é um sistema capaz de fornecer serviço correto mesmo na presença de faltas. Os mecanismos para implementar tolerância a faltas podem ser divididos em (CORREIA, 2005):

- Mascaramento de faltas: usar redundância de componentes para evitar que faltas provoquem uma falha do sistema; e
- Detecção e processamento: detectar a ocorrência de erros e tomar medidas para neutralizá-los.

Evidentemente, não se pode construir um sistema capaz de tolerar um número arbitrário de faltas de qualquer natureza. A semântica de falhas de um sistema ou componente define as maneiras pelas quais este sistema/componente pode falhar, enquanto que a hipótese de faltas de um sistema define o tipo e a frequência das faltas que este deve ser capaz de tolerar (CRISTIAN, 1991). A hipótese de faltas de um sistema depende portanto da semântica de falhas de seus componentes. As semânticas de falhas mais comuns para processos e enlaces de comunicação são mostradas na Tabela 1 (KSHEMKALYANI; SINGHAL, 2011; STEEN; TANENBAUM, 2023). As semânticas mais pertinentes a este trabalho são a de parada, usada para componentes que podem ficar indisponíveis, e a bizantina ou arbitrária (com ou sem autenticação), usada para componentes que podem sofrer uma intrusão. A semântica arbitrária é considerada mais fraca que a de parada, pois adota premissas menos restritivas sobre o comportamento do sistema. A quantidade de componentes necessária para tolerar faltas depende da semântica de falhas adotada. É possível tolerar até f faltas de parada usando $n \geq 2f+1$ componentes, considerando o protocolo de Replicação Máquina de Estados (RME), por exemplo. Faltas arbitrárias, por sua vez, geralmente exigem $n \geq 3f+1$ componentes (LAMPORT; SHOSTAK; PEASE, 1982). Existe portanto um *trade-off*: semânticas de falhas mais fracas costumam exigir mais componentes para tolerar um determinado número f de faltas.

Tabela 1 – Semânticas de falhas mais comuns

Semântica de falhas	**Comportamento**
Falhas de processo	
Parada (*crash*)	Processo para definitivamente, mas funciona corretamente até parar. Os demais componentes não são notificados da parada
Omissão	Processo deixa de responder a requisições (mas não para)
Omissão de recepção	Processo deixa de receber requisições
Omissão de resposta	Processo deixa de enviar respostas
Temporal	Processo responde fora do intervalo de tempo esperado
Resposta	Processo responde incorretamente
Valor	O valor da resposta é incorreto
Transição de estados	Processo realiza uma transição de estado incorreta
Arbitrária (bizantina)	Processo pode gerar respostas arbitrárias em instantes arbitrários
Bizantina com autenticação	Processo pode falhar arbitrariamente, mas se este alega ter recebido uma mensagem de um processo correto, essa alegação pode ser autenticada usando mecanismos criptográficos (*e.g.*, assinatura digital)
Falhas de enlace (ou de comunicação)	
Parada	Enlace para definitivamente de transmitir mensagens, mas funciona corretamente até parar
Omissão	Enlace pode deixar de transmitir algumas mensagens
Arbitrária (bizantina)	Enlace pode se comportar de forma arbitrária, incluindo descarte, alteração ou falsificação de mensagens

Fonte: Elaborado pela autora (2024).

Um conceito relacionado é o de cobertura de premissas (POWELL, 1995). Um sistema projetado para tolerar um determinado tipo de faltas pode falhar caso a semântica de falhas adotada não seja respeitada. Por exemplo, um sistema tolerante a faltas de parada pode—e provavelmente irá—falhar se um componente exibir uma falha arbitrária. Da mesma forma, um sistema projetado para tolerar f faltas irá falhar se $f' > f$ componentes falharem. A cobertura de premissas é a probabilidade de que os componentes do sistema falhem de acordo com a semântica adotada. Tentar reduzir o *overhead* dos mecanismos de TF usando uma semântica de falhas mais forte pode ser inútil caso as premissas adotadas tenham baixa cobertura.

A tolerância a intrusões consiste, como já foi mencionado, em aplicar o paradigma da tolerância a faltas no domínio da segurança, adotando a seguinte perspectiva (VERÍSSIMO; NEVES; CORREIA, 2003):

- Presume-se que o sistema permaneça mais ou menos vulnerável;
- Presume-se que os componentes do sistema podem ser atacados, e que alguns desses ataques serão bem sucedidos; e
- Garante-se que o sistema como um todo não falhe, permanecendo seguro e operacio-

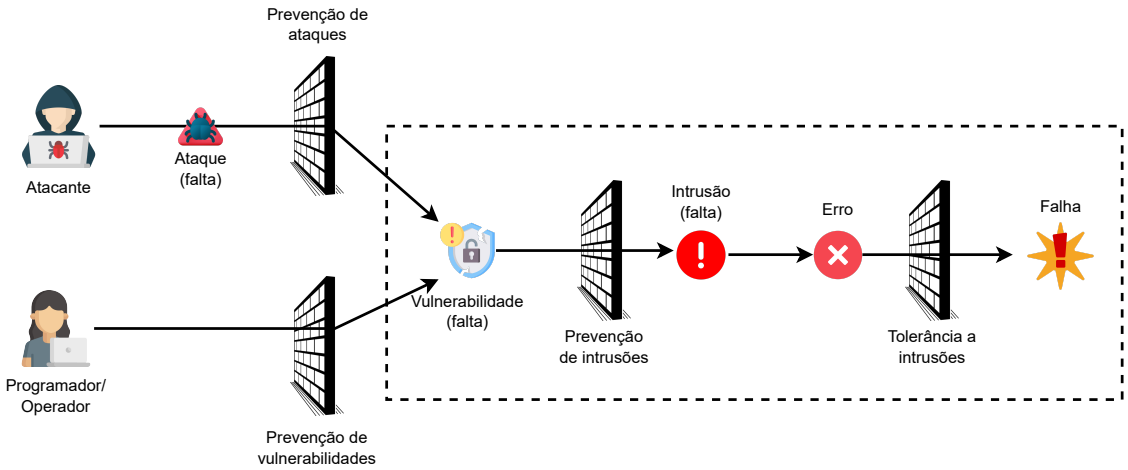
nal, desde que o número de componentes comprometidos permaneça dentro do limite estabelecido em projeto.

A TI considera os conceitos de ataque, vulnerabilidade e intrusão, tradicionais em segurança (SHIREY, 2007), como diferentes tipos de faltas (CORREIA, 2005):

- Uma vulnerabilidade é uma falta de projeto ou de configuração, geralmente acidental (*i.e.*, não intencional), que pode ser explorada com fins maliciosos;
- Um ataque é uma falta intencional, maliciosa, que visa explorar uma ou mais vulnerabilidades; e
- Uma intrusão é o resultado de um ataque que obtém sucesso na exploração de uma ou mais vulnerabilidades, e que pode levar à falha do sistema.

Essa relação entre ataque, vulnerabilidade e intrusão, ilustrada pela Figura 2, permite entender o processo de falha de um sistema e, por conseguinte, os meios que podem ser usados para evitar falhas. A segurança tradicional está focada na prevenção, seja de ataques (*e.g.*, usando um *firewall* para filtrar tráfego de rede), de vulnerabilidades (*e.g.*, encontrando e removendo vulnerabilidades em código fonte, ou usando linguagens de programação mais seguras) ou de intrusões (*e.g.*, detectando tráfego de rede malicioso e derrubando a conexão). A TI não diminui a importância da prevenção, mas admite que esta nem sempre será bem sucedida, e arquiteta o sistema de modo a permitir que este continue funcionando corretamente caso ocorram intrusões.

Figura 2 – O modelo AVI e os mecanismos para evitar falhas



Fonte: Adaptado de Correia (2005).

Arquiteturas tolerante a intrusões geralmente pressupõem sistemas distribuídos, e fazem uso de diversos blocos de construção (VERÍSSIMO; NEVES; CORREIA, 2003; CORREIA, 2005):

- **Replicação**: consiste em distribuir cópias do código e dos dados de determinado serviço por um conjunto de servidores (CORREIA, 2005). A replicação é fundamental para remover do sistema pontos únicos de falha, ou seja, componentes que comprometeriam a segurança do sistema em caso de falha ou intrusão.

  Uma abordagem usual é a Replicação Máquina de Estados (RME) ou *State Machine Replication* (SMR) (SCHNEIDER; ZHOU, 2005), a qual é explicada em mais detalhes na Seção 2.2.

- **Diversidade**: consiste em usar diferentes implementações de um mesmo sistema (ou subsistema) para obter tolerância a faltas, partindo da premissa que projetos e implementações independentes apresentem falhas de software também independentes (OBELHEIRO; BESSANI; LUNG, 2005). A diversidade reduz a probabilidade de que várias réplicas de um sistema TI sejam comprometidas simultaneamente, por apresentarem vulnerabilidades comuns.

  Há duas considerações importantes a se fazer sobre a diversidade. A primeira é que um sistema computacional possui vários componentes que podem ser diversificados (ou eixos de diversidade), desde o hardware, passando por sistema operacional, bibliotecas, suportes de execução, e chegando até a aplicação. A diversidade pode ser usada de forma parcial: diversificar alguns eixos já oferece uma maior proteção contra vulnerabilidades comuns do que não diversificar nenhum deles. A segunda consideração é que a diversidade não se contrapõe necessariamente à replicação: é possível ter diferentes implementações de uma aplicação desde que todas as réplicas sigam a mesma especificação.

- **Criptografia de limiar**: chaves criptográficas também podem constituir um ponto único de falha. Voltando ao exemplo da autoridade certificadora tolerante a intrusões, se cada réplica pudesse emitir sozinha um certificado, então a arquitetura distribuída, ainda que melhorasse a disponibilidade, pioraria a integridade do sistema, pois bastaria comprometer uma das n réplicas para conseguir forjar certificados. A criptografia de limiar (*threshold cryptography*) (GEMMELL, 1997; BRANDÃO; MOUHA; VASSILEV, 2019) minimiza este problema, permitindo que uma chave criptográfica seja particionada em frações que são disseminadas pelos nós de um sistema distribuído de modo que *(i)* seja possível reconstituir a chave (ou executar uma função criptográfica, como uma assinatura digital) usando apenas um subconjunto das frações e *(ii)* a indisponibilidade de parte das frações não comprometa a disponibilidade da chave ou função.

Embora tenha estado em maior evidência no início dos anos 2000, a tolerância a intrusões permanece uma área ativa de pesquisa. Isso é evidenciado por diversos trabalhos sobre arquiteturas TI, para uma variada gama de aplicações, que foram publicados na última década, tais como (a lista pretende ser ilustrativa, e não exaustiva):

- Serviços de autenticação e autorização (BARRETO et al., 2013; KREUTZ et al., 2014; BARRETO; FRAGA; SIQUEIRA, 2021);
- Serviços de armazenamento em nuvem (BESSANI et al., 2013; MADAN et al., 2016);
- Plataforma MapReduce (COSTA et al., 2016);
- Redes *overlay* (OBENSHAIN et al., 2016; TRESTIOREANU et al., 2021);
- Sistemas de direção autônoma (VOELP; VERÍSSIMO, 2018);
- Sistemas SCADA (*Supervisory Control and Data Acquisition*) (BABAY et al., 2019);
- Arquiteturas de microsserviços (FLORA, 2020);
- Proteção de *workflows* científicos (WANG et al., 2020);
- Serviços de *single sign-on* (SSO) (BAUM et al., 2020; MAGNANINI; FERRETTI; COLAJANNI, 2021); e
- Detecção de *fake news* (SHAN et al., 2021).

2.2 REPLICAÇÃO MÁQUINA DE ESTADOS (RME)

A Replicação Máquina de Estados (RME) (SCHNEIDER, 1990) é uma abordagem usual na implementação de sistemas tolerantes a intrusões (SCHNEIDER; ZHOU, 2005), e desempenha um papel central na proposta deste trabalho. Na RME, um servidor oferece um conjunto de operações a seus clientes, que enviam requisições ao servidor e esperam por respostas. Um servidor é estruturado como uma máquina de estados determinística, definida por um conjunto de variáveis de estado que são modificadas apenas em função das requisições recebidas dos clientes. Réplicas desse servidor executam em nós distintos de um sistema distribuído e devem atender a três propriedades (SCHNEIDER, 1990):

(i) partir do mesmo estado;

(ii) executar as mesmas solicitações na mesma ordem; e

(iii) atingir o mesmo estado final.

Para garantir *(i)*, é necessário inicializar todas as réplicas com o mesmo estado, o que pode ser complexo quando são ativadas novas réplicas ou quando uma réplica faltosa é recuperada e precisa retornar ao conjunto de réplicas ativas (BESSANI; ALCHIERI, 2014). Já a garantia da propriedade *(ii)* envolve o uso de um protocolo de difusão atômica (ou difusão com ordem total) (HADZILACOS; TOUEG, 1994), pelo qual todos os processos corretos de um determinado grupo devem entregar todas as mensagens enviadas a esse grupo na mesma ordem. Por fim, para garantir *(iii)*, todas as operações realizadas pelas réplicas devem ser determinísticas (determinismo de

réplica). Assumindo que esses três requisitos sejam cumpridos, uma implementação RME deve satisfazer as seguintes propriedades de *safety* e *liveness* (BESSANI; ALCHIERI, 2014):

- *Safety*: todas as réplicas corretas executam a mesma sequência de operações; e
- *Liveness*: todas as operações de clientes corretos são executadas.

Uma forma comum de implementar difusão atômica é usando um protocolo de consenso para que as réplicas entrem em acordo sobre quais mensagens serão entregues à aplicação, e em que ordem (DÉFAGO; SCHIPER; URBÁN, 2004). Essa estratégia se baseia na equivalência entre os problemas de difusão atômica e consenso (HADZILACOS; TOUEG, 1994).

Existem protocolos para replicação máquina de estados para diferentes semânticas de falhas, sendo as mais comuns as de parada e bizantinas (Tabela 1, página 20). O número n de réplicas necessárias para tolerar f faltas é diferente em cada caso; a maioria dos protocolos requer $n \geq 2f + 1$ réplicas para faltas de parada e $n \geq 3f + 1$ réplicas para faltas bizantinas (BESSANI; ALCHIERI, 2014).

2.3 CONCEITOS DE SPIFFE/SPIRE

A presente seção tem como objetivo apresentar o que é o SPIFFE, focando nos seus elementos principais (Seção 2.3.1), bem como explicar o funcionamento da arquitetura do SPIRE e a funcionalidade dos seus componentes (Seção 2.3.2).

2.3.1 SPIFFE

O SPIFFE (*Secure Production Identity Framework for Everyone*) é um conjunto de especificações de código aberto que visa prover gerenciamento seguro de identidades de software em ambientes dinâmicos e heterogêneos (FELDMAN et al., 2020). O SPIFFE faz uso de alguns conceitos básicos, são eles: Carga de Trabalho (*Workload*), Domínio de Confiança (*Trust Domain*), ID SPIFFE, *SPIFFE Verifiable Identity Document* (SVID), *Workload API*, Pacote de Confiança (*Trust Bundle*), e Federação de Domínios.

No SPIFFE, uma Carga de Trabalho nada mais é do que um software, o qual é implantado com uma configuração específica para uma única finalidade (SPIFFE, 2019b). Por exemplo, um servidor *Web*, uma instância de um banco de dados, entre outros. Todas as Cargas de Trabalho identificadas no mesmo Domínio de Confiança, o qual refere-se à raiz de confiança de um sistema, recebem documentos de identidade que podem ser verificados em relação às chaves raiz desse domínio.

Cada Carga de Trabalho é identificada por um ID SPIFFE, um *Uniform Resource Identifier* (URI) no formato `spiffe://domínio_de_confiança/id_da_carga`. Um domínio de confiança pode representar um indivíduo, organização, ambiente ou departamento executando sua própria infraestrutura SPIFFE independente. O implementador é livre para escolher qualquer

nome de domínio que considere adequado, pois não há autoridade centralizada para regulamentação ou registro de nomes de domínios de confiança (SPIFFE, 2022b). Para evitar que dois domínios selecionem nomes idênticos, aconselha-se a seleção de nomes com alta probabilidade de serem globalmente exclusivos. Em casos de colisão, os domínios não poderão conectar-se entre si, mas continuarão a operar de forma independente.

O implementador também é livre para definir o caminho (ID da carga) de um ID SPIFFE. Os caminhos podem ser hierárquicos, *i.e.*, semelhantes a nomes de caminhos em sistemas de arquivos (FELDMAN et al., 2020). O implementador pode identificar serviços diretamente pela funcionalidade que eles representam do ponto de vista da aplicação e do ambiente em que ela é executada (SPIFFE, 2022b). Por exemplo, o ID SPIFFE `spiffe://teste.example.com/pagamentos/mysql` pode referir-se ao banco de dados `mysql` de um serviço de `pagamentos` executado em um ambiente de `teste`. O implementador também pode identificar proprietários de serviços, como orquestradores e plataformas de nível superior que têm seus próprios conceitos de identidade (*e.g.*, contas de serviço Kubernetes) (SPIFFE, 2022b). Por exemplo, o ID SPIFFE `spiffe://k8s-west.example.com/ns/teste/sa/padrao` pode representar um cenário em que o administrador de `example.com` está executando um *cluster* Kubernetes `k8s-west.example.com`, que possui um *namespace* de `teste`, e dentro deste há uma conta de serviço (`sa`) chamada `padrao`. Por fim, o implementador pode optar por usar um ID SPIFFE opaco (sem nenhuma informação hierárquica visível), mantendo os metadados associados em um banco de dados secundário (FELDMAN et al., 2020). Um exemplo de ID opaco seria `spiffe://example.com/3576-98573-9afcd`.

Um SVID é um documento de identidade criptograficamente verificável usado para provar a identidade de uma Carga de Trabalho. Um SVID contém um único ID SPIFFE. Os formatos atualmente suportados são: certificados X.509 e *tokens* JWT (*JavaScript Web Token*). Um Domínio de Confiança representa uma parte do espaço de nomes de IDs SPIFFE sob autoridade de um conjunto específico de chaves públicas. Esse conjunto de chaves é o Pacote de Confiança. Todo Domínio de Confiança SPIFFE tem um pacote associado a ele, e o material desse pacote é usado para validar SVIDs que afirmam residir no referido Domínio de Confiança. O nome do domínio que o pacote representa deve ser registrado ao armazenar ou gerenciar pacotes SPIFFE. Esse registro deve ser nominal, por meio do uso de uma tupla `<nome_domínio_de_confiança, pacote>`. Devido a isso, ao validar um SVID é selecionado o pacote correspondente ao domínio em que o SVID reside.

Uma Carga de Trabalho obtém seu SVID e o pacote de confiança do seu domínio por meio da *Workload API* oferecida por um Agente (FALCÃO et al., 2022). Para evitar que uma Carga use uma credencial para se identificar, o Agente realiza a atestação de carga: informações do sistema operacional e da plataforma de nuvem/contêineres a respeito da Carga são usadas para identificá-la automaticamente e selecionar o SVID correto. Quando o pacote de confiança ou os SVIDs são rotacionados, o que ocorre periodicamente, as Cargas são notificadas para que possam obter as versões mais atualizadas. Uma Carga de Trabalho valida os SVIDs recebidos

usando as chaves públicas do Pacote de Confiança.

Um mecanismo de federação de domínios permite que uma Carga de Trabalho em um Domínio de Confiança valide SVIDs emitidos em outros domínios, possibilitando a comunicação segura entre serviços em domínios distintos. A federação de domínios no contexto do SPIRE é discutida na Seção 2.3.2.3.

2.3.2 SPIRE

O SPIRE (*SPIFFE Runtime Environment*) (SPIFFE, 2019c) é a implementação de referência do SPIFFE. O SPIRE executa atestação de nós e Cargas de Trabalho para, com base em um conjunto predefinido de condições, emitir e verificar SVIDs de Cargas de Trabalho. A Seção 2.3.2.1 apresenta a arquitetura do SPIRE. Na Seção 2.3.2.2 é descrito o processo de atestação realizado pelo Servidor e Agentes. Por fim, a Seção 2.3.2.3 apresenta as arquiteturas de implantação que têm como objetivo melhorar a disponibilidade do Servidor SPIRE.

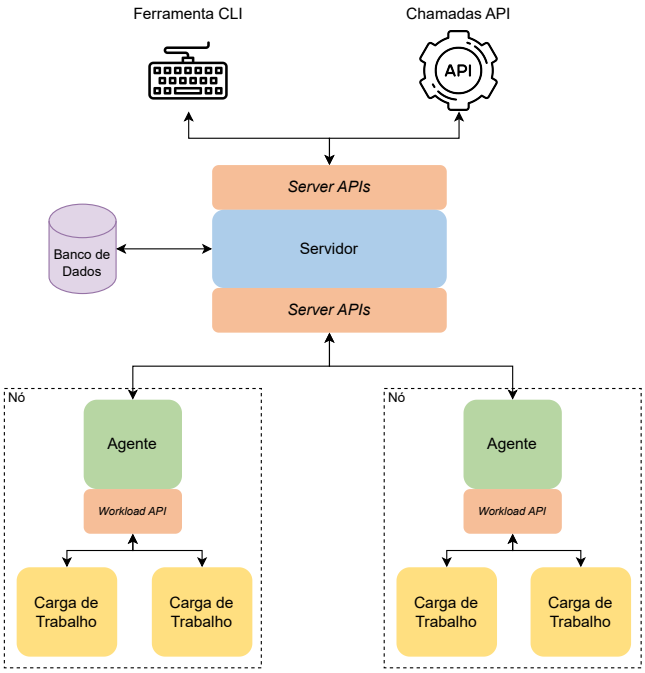
2.3.2.1 Arquitetura

A arquitetura do SPIRE consiste em dois componentes principais, o Servidor e o Agente. Uma implementação SPIRE pode ter um ou mais Agentes. Esta arquitetura pode ser observada na Figura 3. Cada Agente executa em um nó (que pode ser uma máquina física ou virtual), e serve as Cargas de Trabalho que executam nesse nó, disponibilizando para isso a *Workload API*. O Agente é responsável por atestar Cargas de Trabalho, solicitar ao Servidor os SVIDs das Cargas alocadas ao seu nó, e fornecer SVIDs às suas Cargas quando demandado. O Agente mantém ainda um *cache* das entradas e dos SVIDs obtidos junto ao Servidor. Para realizar o gerenciamento desse *cache* o Agente possui um *Manager*, o qual é responsável por *(i)* solicitar ao Servidor as entradas autorizadas, *(ii)* atualizar o *cache* e *(iii)* solicitar os SVIDs para as devidas entradas. Esse conjunto de tarefas é executado periodicamente dentro de um intervalo de tempo predefinido. Este texto refere-se a este *loop* como "processo de sincronização".

O Servidor é responsável por armazenar chaves de assinatura, emitir SVIDs para Cargas de Trabalho, atestar os Agentes e fornecer-lhes os SVIDs das suas respectivas Cargas. Um resumo dos fluxos da troca de informações entre Agente e Servidor pode ser observado na Figura 4. Embora a documentação do SPIRE ainda mencione a existência da *Registration API* e da *Node API*, ambas foram reorganizadas em múltiplas APIs, as quais são mencionadas neste texto como *Server APIs*. A *Registration API* e a *Node API* foram oficialmente descontinuadas na versão 0.12.0, em dezembro de 2020 (SPIRE, 2020).

A interação de um Agente com o Servidor ocorre por meio de *Server APIs*, antes conhecidas como *Node API*. As Cargas de Trabalho gerenciadas pelo Servidor são informadas por meio de entradas de registro, que são manipuladas usando *Server APIs* (anteriormente conhecidas como *Registration API*) ou uma interface de linha de comando (*command-line interface*, CLI). Cada entrada de registro possui um ID pai (*Parent ID*), que informa onde uma

Figura 3 – Arquitetura SPIRE



Fonte: Adaptado de SPIFFE (2019c).

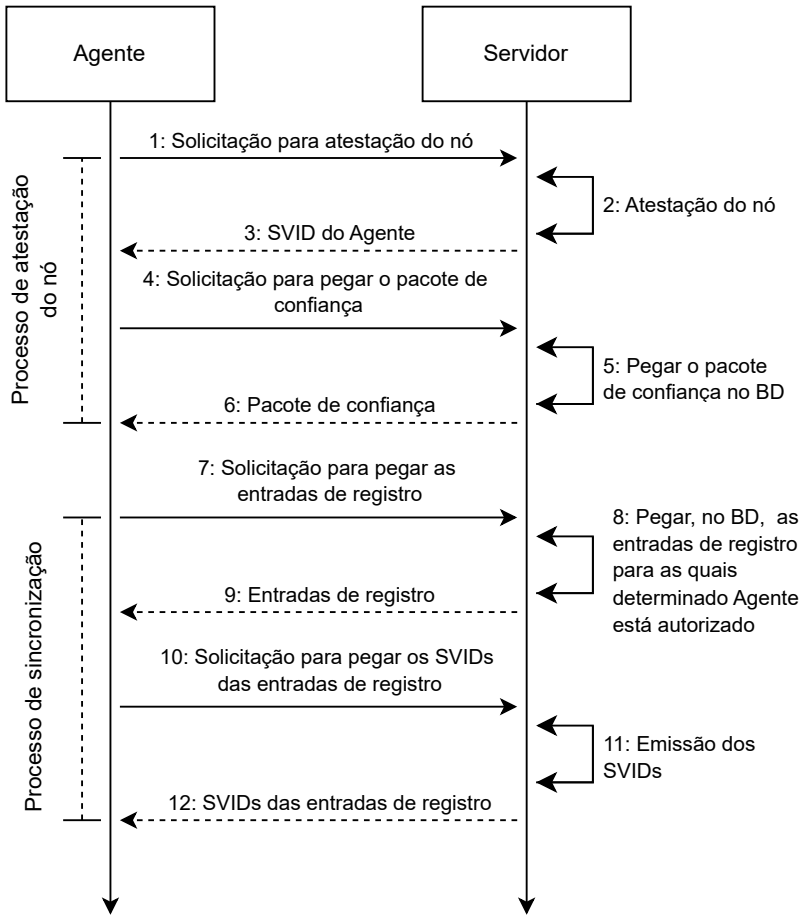
determinada Carga de Trabalho ou nó deve ser executada, um ID SPIFFE, e informações que ajudem a identificar a Carga de Trabalho ou nó (FELDMAN et al., 2020). Em outras palavras, as entradas de registro especificam seletores que determinam as condições sob as quais o SVID para um determinado ID SPIFFE deve ser emitido (SPIFFE, 2019c). As chaves de assinatura (que constituem o pacote de confiança), os SVIDs emitidos pelo Servidor e as entradas de registro são armazenadas em um banco de dados (BD), que não é considerado parte da arquitetura SPIRE mas desempenha nesta um papel chave, como discutido na Seção 2.3.2.3.

O SPIRE é implementado na linguagem Go. As *Server APIs* e a *Workload API* são expostas como servidores gRPC, um sistema de código aberto de chamada de procedimento remoto (RPC, *Remote Procedure Call*) que pode ser usado com diversas linguagens de programação e sistemas operacionais (GRPC, 2024). O gRPC usa *Protocol Buffers* (BUFFERS, 2024) para serializar e desserializar dados estruturados, o que permite que clientes e servidores gRPC sejam implementados em linguagens diferentes, bem que executem em plataformas distintas.

2.3.2.2 Atestação

A atestação é o processo de provar com certeza a identidade de uma Carga de Trabalho ou do nó do Agente, por meio das informações disponíveis como evidência (FELDMAN et al., 2020). Existem dois tipos de atestação no SPIRE: atestação de nó e atestação de Carga de Trabalho.

Figura 4 – Fluxo de informação entre Servidor e Agente



Fonte: Elaborado pela autora (2024).

A atestação de nó ocorre quando um Agente é executado pela primeira vez. Aqui são declarados atributos que descrevem os nós (seletores) (FELDMAN et al., 2020). O Agente e o Servidor são responsáveis por verificar a identidade do nó no qual o Agente está sendo executado. Estes realizam a verificação por meio de *plugins* conhecidos como atestadores de nó (SPIFFE, 2019c). Com o objetivo de provar a identidade do nó, os atestadores interrogam o nó e seu ambiente em busca de informações que somente aquele nó deve possuir. Se a atestação for bem sucedida, o Servidor emite um ID SPIFFE exclusivo para o Agente em questão.

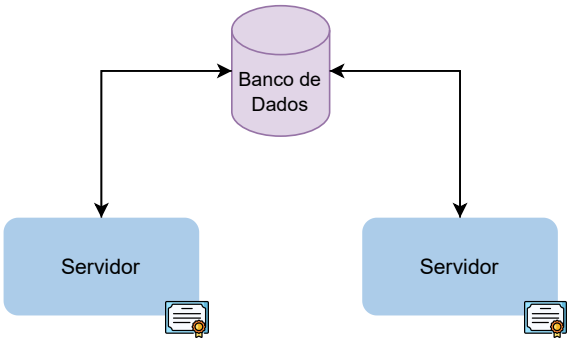
Já a atestação de Cargas de Trabalho tem como objetivo determinar a identidade da Carga que estabeleceu uma conexão com a API disponibilizada pelo Agente. Após essa conexão, o Agente interroga o *kernel* do nó para obter os seletores da Carga de Trabalho. Em seguida, o Agente determina a identidade da Carga de Trabalho comparando os seletores descobertos com as entradas de registro e, se a atestação for bem sucedida, retorna o SVID correto para a Carga de Trabalho (SPIFFE, 2019c).

O SPIRE é capaz de atestar as identidades de Cargas de Trabalho que executam em máquinas (nós) físicas ou virtuais as quais usam Linux (SPIFFE, 2019a). No processo de atestação é necessário que o Servidor SPIRE estabeleça confiança com um Agente, o qual está em execução em um nó Linux. Essa confiança pode ser estabelecida por meio de três métodos: *Join Token*, Certificado X.509, e Certificado SSH. O método *Join Token* consiste no Servidor gerar um *token* de uso único, o qual é fornecido ao Agente. Quando o Agente é iniciado, este apresenta o *token*. O Servidor validará o *token* e emitirá um SVID ao Agente. Nas inicializações subsequentes, o Agente usará esse SVID para se autenticar ao Servidor. Outra possibilidade de autenticação é por meio da validação de um certificado X.509 folha, o qual é exclusivo e foi previamente instalado no nó. Esse certificado folha é gerado a partir de uma única chave e certificados comuns. O Servidor deve possuir a chave raiz e eventuais certificados intermediários para validar o certificado folha apresentado. Por fim, um nó pode ser automaticamente provisionado com um certificado SSH, o qual é exclusivo. Esse certificado pode ser usado para autenticar o nó.

2.3.2.3 Arquiteturas de Implantação

O SPIRE oferece três arquiteturas de implantação que melhoram a disponibilidade do Servidor, que são: SPIRE-HA, Nested SPIRE e SPIRE Federado. A primeira delas, SPIRE-HA, é mostrada na Figura 5. Ela consiste em replicar o Servidor, com todas as réplicas acessando um BD centralizado (SPIFFE, 2021). As réplicas são funcionalmente idênticas: cada uma delas pode gerenciar entradas de registro, emitir SVIDs, e interagir com Agentes. Em outras palavras, todos os Servidores estão no mesmo domínio de confiança, acessando o mesmo BD compartilhado. O banco de dados fica responsável por armazenar as chaves de assinatura, os SVIDs emitidos e as entradas de registro para todos os Servidores. Isso permite que, enquanto houver uma réplica funcionando corretamente, o Servidor permaneça operacional, sujeito à disponibilidade do BD.

Figura 5 – Arquitetura SPIRE-HA

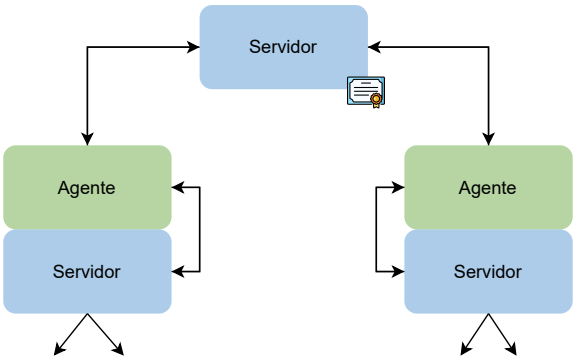


Fonte: Adaptado de SPIFFE (2021).

Já a arquitetura Nested SPIRE (Figura 6) usa uma hierarquia de Servidores SPIRE como autoridades certificadoras (ACs), sendo uma AC raiz e múltiplas ACs intermediárias (SPIFFE,

2021). Em outras palavras, o Servidor de nível superior pode ser representado como um servidor global, e os Servidores *downstream* como servidores regionais. No caso de um Servidor falhar, o processo para reconfigurar os Agentes afetados para outro Servidor é complexo. É necessário modificar a entrada de registro do nó do Agente, dado que é o Servidor que está realizando a atestação do nó, as entradas do nó têm seu ID pai definido como o ID SPIFFE do Servidor (FELDMAN et al., 2020).

Figura 6 – Arquitetura Nested SPIRE



Fonte: Adaptado de SPIFFE (2021).

A arquitetura do SPIRE Federado permite que uma Carga de Trabalho em um determinado domínio de confiança autentique uma Carga que está em um domínio diferente. Quando dois Domínios de Confiança A e B são federados (Figura 7), o Servidor do domínio A precisa ser configurado para obter o Pacote de Confiança do domínio B (e vice-versa). Em linhas gerais, isso envolve os seguintes passos (SPIFFE, 2022a):

1. O Servidor B define um *endpoint*, que é uma URL por meio da qual este disponibiliza o Pacote de Confiança do domínio B;

2. O Servidor A é configurado com a informação de que ele está federado com o domínio B, cujo *endpoint* é aquele definido no passo 1;

3. O Servidor A pode autenticar o *endpoint* do domínio B de duas formas, dependendo de como esse *endpoint* foi definido no passo 1:

    a) Se for Web PKI (BRAUN et al., 2014), o *endpoint* possui um certificado X.509, assinado por uma AC da Web PKI, associado a seu nome DNS. Se o certificado for válido para o *endpoint*, este é considerado autêntico.

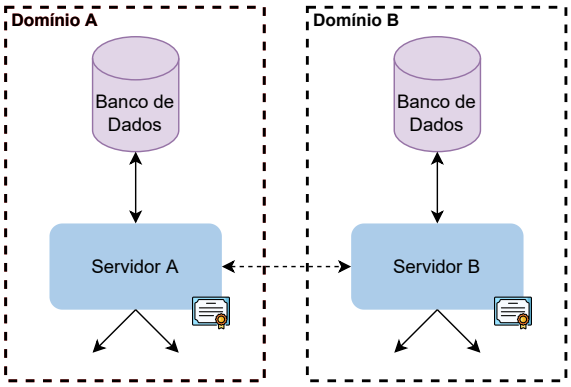
    b) Se for SPIFFE, o Servidor A precisa ser alimentado com um Pacote de Confiança inicial que permita validar o SVID do *endpoint* do Servidor B. Tipicamente, esse pacote é o Pacote de Confiança do domínio B válido no momento da inicialização, o qual precisa ser obtido *off-line* com o administrador do domínio B. Se o SVID

apresentado puder ser validado com as chaves deste Pacote de Confiança, o *endpoint* é considerado autêntico.

Uma vez autenticado o *endpoint* do domínio B, futuras atualizações do Pacote de Confiança desse domínio poderão ser automaticamente validadas pelo Servidor do domínio A.

Procedimento análogo deve ser feito para que o Servidor do domínio B possa receber e validar o Pacote de Confiança do domínio A. Para que uma Carga de Trabalho no domínio A possa validar IDs SPIFFE do domínio B, o Servidor A envia os Pacotes de Confiança dos domínios federados para seus Agentes, que por sua vez repassam esses pacotes para suas Cargas de Trabalho.

Figura 7 – Arquitetura SPIRE Federado



Fonte: Adaptado de SPIFFE (2021).

O SPIRE Federado também permite a interação de um domínio SPIFFE com sistemas externos compatíveis com SPIFFE; por exemplo, a interação entre uma implantação SPIRE e uma malha de serviço Istio[1] (SPIFFE, 2021).

Do ponto de vista de tolerância a faltas, as três arquiteturas de implantação apresentadas nesta seção são projetadas para lidar com faltas de parada. A arquitetura SPIRE-HA tolera faltas do Servidor no contexto de um único Domínio de Confiança. Embora isto não faça parte da arquitetura, seria possível tolerar faltas de parada também do banco de dados adotando um BD replicado. A arquitetura Nested SPIRE particiona as Cargas de Trabalho de um Domínio de Confiança em um conjunto de Servidores, de acordo com a organização topológica dessas Cargas. O objetivo é que a parada de um Servidor afete apenas um subconjunto de Agentes e Cargas. A arquitetura SPIRE Federado considera múltiplos domínios de confiança, e permite que a parada do Servidor de um domínio afete apenas as Cargas nesse domínio. Nenhuma das três arquiteturas é capaz de lidar com o comprometimento da segurança de um Servidor, cujas consequências são discutidas na Seção 3.1.

[1] Uma malha de serviço é uma camada de infraestrutura para aplicativos baseados em microsserviços, os quais são adequados especificamente para arquiteturas de aplicativos distribuídos (KOSCHEL et al., 2021). O Istio (<https://istio.io>) é uma malha de serviço de código aberto que fornece uma maneira eficiente de proteger, conectar e monitorar serviços.

2.4 TRABALHOS RELACIONADOS

Nesta seção são identificados e discutidos os principais trabalhos que abordam as duas questões principais relacionadas a esta dissertação. A Seção 2.4.1 apresenta trabalhos relacionados a tecnologia SPIFFE/SPIRE. Já os trabalhos relacionados a tolerância a intrusões para serviços de autenticação são apresentados na Seção 2.4.2.

2.4.1 SPIFFE/SPIRE

SPIFFE e SPIRE são tecnologias recentes, ainda pouco exploradas na literatura. Uma experiência relevante é Falcão et al. (2022), que integra o SPIRE com a tecnologia Intel SGX para computação confidencial. Os autores apresentam um *plugin* SPIRE que emite identidades para microsserviços confidenciais, *i.e.*, atesta e fornece identidades para cargas de trabalho SGX. A proposta permite a interoperabilidade entre cargas de trabalho regulares e confidenciais. O artigo explora ainda o desempenho da execução dos componentes SPIRE em enclaves SGX, observando um baixo *overhead*.

Jensen (2022) apresenta uma modelagem de ameaças usando a abordagem STRIDE para uma aplicação que utiliza o SPIFFE para permitir acesso de uma carga de trabalho a um banco de dados. Entretanto, as ameaças encontradas não vão além das ameaças apresentadas, de forma dispersa, na documentação do SPIFFE (FELDMAN et al., 2020). O trabalho ainda apresenta quatro cenários avaliados e testados: acesso às APIs das cargas de trabalho, roubo de chaves, DoS no *endpoint* das cargas, e vulnerabilidade de uma aplicação. Contudo, alguns ataques foram bem sucedidos devido ao uso de um ambiente Kubernetes inseguro. As mitigações propostas envolvem modificações nas configurações da máquina, não necessariamente no *framework* SPIFFE.

Pontes et al. (2023) apresentam um *plugin* SPIRE para atestação de nó. Este faz uso dos recursos do AMD SEV-SNP e do *kernel* do Linux para atestar máquinas virtuais confidenciais. O artigo ainda analisa o tempo que o agente SPIRE leva para inicializar e ficar pronto para atender as cargas de trabalho. Aqui observa-se um *overhead* de 112.5%. Embora este trabalho se preocupe com uma das principais operações do Servidor SPIRE (a atestação de nó), o Servidor continua sendo um componente crítico na arquitetura.

Jessup et al. (2024) apresentam um caso de uso que explora a delegação de credenciais externas, no qual um novo documento denominado *Delegated Assertion SVID* (DVID) é emitido. Este documento implementa o uso da prova de conhecimento zero, e com ele é possível identificar o usuário final que fez a requisição original. Entretanto, o servidor considerado na proposta representa um ponto único de falha na arquitetura do SPIFFE. Já Cochak et al. (2024) visam continuar o desenvolvimento do documento DVID apresentado em Jessup et al. (2024), bem como empregar um esquema de aninhamento de *tokens*, o qual suporta assinatura distribuída e um mecanismo de rastreamento do fluxo por quais serviços o *token* transitou. Contudo, o trabalho não considera o uso de um servidor tolerante a intrusões.

Observa-se na Tabela 2 que, dos quatro componentes principais do SPIRE (Agente, BD, Carga de Trabalho, e Servidor), os trabalhos revisados se concentram na proposição de melhorias relacionadas às Cargas de Trabalho. O único trabalho cuja preocupação envolve o Servidor SPIRE é Pontes et al. (2023), que tem como foco a atestação de máquinas virtuais confidenciais, ou seja, o Servidor continua representando um ponto único de falha na arquitetura do SPIFFE/SPIRE.

Tabela 2 – Comparação entre os trabalhos que abordam a tecnologia SPIFFE/SPIRE

Referência	Agente	Banco de Dados	Carga de Trabalho	Servidor
Cochak et al. (2024)	Não	Não	Sim	Não
Falcão et al. (2022)	Não	Não	Sim	Não
Jensen (2022)	Não	Não	Sim	Não
Jessup et al. (2024)	Não	Não	Sim	Não
Pontes et al. (2023)	Não	Não	Não	Sim
Este trabalho	Não	Sim	Não	Sim

Fonte: Elaborado pela autora (2024).

2.4.2 Tolerância a Intrusões

Existem experiências relacionadas no contexto de tolerância a intrusões para serviços de autenticação, incluindo arquiteturas tolerantes a intrusões para OpenID (OPENID, 2023) e Radius (RIGNEY et al., 2000; IBM, 2023). Barreto et al. (2013) têm como objetivo garantir o comportamento correto na autenticação em grandes sistemas por meio do desenvolvimento de um provedor de identidades OpenID tolerante a intrusões. Este é baseado na tecnologia de virtualização, as máquinas virtuais e componentes confiáveis são isolados e se comunicam por meio de uma memória compartilhada. Entretanto, o *Agreement Service* e o Servidor de Autenticação são pontos únicos de falha da arquitetura proposta. Além disso, o sistema não tolera falhas físicas (*e.g.*, apagões de energia) e lógicas (*e.g.*, falhas de software). Em Barreto, Fraga e Siqueira (2021) os autores apresentam uma abordagem que pode ser usada de forma complementar à citada em Barreto et al. (2013). Esta faz uso de provedores de *cloud* para armazenar as credenciais e atributos dos usuários. As modificações são efetuadas no sentido de integrar os algoritmos de compartilhamento de segredo (*e.g.*, VSS e PVSS) à arquitetura do serviço de autenticação tolerante a intrusão. Contudo, os pontos únicos de falhas citados anteriormente permanecem.

Já Kreutz et al. (2014) propõem uma arquitetura de resiliência para melhorar a segurança e confiabilidade das infraestruturas de autenticação e autorização baseadas em Radius e OpenID. Essa arquitetura é baseada no uso de replicação tolerante a intrusões, componentes confiáveis bem definidos, e *gateways* não confiáveis. A arquitetura proposta permite o uso de 1 até $3f_R + 1$ componentes confiáveis, sendo f_R o número máximo de faltas simultâneas toleradas pelas

réplicas de autenticação. Os autores presumem que os componentes confiáveis falham apenas por travamento. Baseado no trabalho anterior, Kreutz et al. (2016) apresentam um conjunto de artefatos de *design* de sistema e uma arquitetura funcional para projetar, implementar e implantar serviços de autenticação e autorização robustos e confiáveis.

Outros trabalhos apresentam provedores de identidade (*identity providers*, IdPs) tolerantes a intrusões em esquemas de login único (*single sign-on*, SSO). A ideia geral é proteger o usuário contra servidores comprometidos, introduzindo um protocolo criptográfico que permite que o usuário se autentique via senha em um conjunto de servidores e obtenha um *token* que pode ser verificado por um provedor de serviço (*service provider*, SP). Magnanini, Ferretti e Colajanni (2021) apresentam um protocolo SSO tolerante a intrusões que emprega uma arquitetura distribuída de n servidores de identidade que autenticam usuários e emitem *tokens* parciais usando assinaturas digitais convencionais (*i.e.*, sem criptografia de limiar). $k \leq n$ *tokens* parciais corretos são combinados do lado do usuário para gerar um *token* que pode ser validado por um provedor de serviço usando um procedimento de verificação específico para o *token* combinado. Isso permite tolerar intrusões em k dos n servidores. A proposta permite que o limiar k seja reconfigurado dinamicamente pelo provedor de serviço, o que proporciona flexibilidade.

Ferretti et al. (2021) apresentam uma nova arquitetura de confiança zero que pode garantir o nível de segurança necessário para ambientes de computação em nuvem. A arquitetura proposta distribui a confiança entre vários componentes do plano de controle para que a arquitetura possa tolerar invasões e se recuperar de ataques bem sucedidos. Para isso os autores realizam a aplicação de técnicas de tolerância a falhas bizantinas (*Byzantine Fault Tolerance*, BFT) na construção de alguns componentes, bem como adotam o protocolo SSO tolerante a intrusões apresentado em Magnanini, Ferretti e Colajanni (2021), e aplicam a distribuição de confiança entre componentes com base na remoção de intermediários, estendendo cada recurso com um *gateway*.

A Tabela 3 resume as experiências relacionadas no contexto de tolerância a intrusões para serviços de autenticação, abrangendo arquiteturas TI para OpenID, Radius, protocolo SSO, e SPIFFE. Observa-se que nenhum trabalho propõe implementar o conceito de tolerância a intrusões para a arquitetura do SPIFFE.

Tabela 3 – Comparação entre as principais soluções identificadas

Referência	OpenID	Radius	*single sign-on*	SPIFFE
Barreto et al. (2013)	Sim	Não	Não	Não
Barreto, Fraga e Siqueira (2021)	Sim	Não	Não	Não
Ferretti et al. (2021)	Não	Não	Sim	Não
Kreutz et al. (2014)	Sim	Sim	Não	Não
Kreutz et al. (2016)	Sim	Sim	Não	Não
Magnanini, Ferretti e Colajanni (2021)	Não	Não	Sim	Não
Este trabalho	Não	Não	Não	Sim

Fonte: Elaborado pela autora (2024).

2.5 CONSIDERAÇÕES PARCIAIS

A tolerância a intrusões em um sistema distribuído tem como objetivo eliminar ou reduzir os pontos únicos de falha na arquitetura. Conforme observado na Figura 3, o SPIRE possui quatro componentes principais: Servidor, Agente, BD, e Cargas de Trabalho. A Seção 3.1 identificará quais são os componentes críticos da arquitetura do SPIRE, e o impacto de violações de disponibilidade e integridade desses componentes.

Apesar do SPIFFE e SPIRE serem ainda pouco explorados na literatura, existem alguns estudos relacionados no contexto de tolerância a intrusões. Portanto, as experiências relatadas na Seção 2.4.2 fornecerão subsídios para a definição de uma versão tolerante a intrusões para o Servidor SPIRE.

3 IT-SPIRE, UM SERVIDOR SPIRE TOLERANTE A INTRUSÕES

Este capítulo apresenta o IT-SPIRE, um Servidor SPIRE tolerante a intrusões. Primeiramente, a Seção 3.1 identifica os componentes críticos do SPIRE e descreve como o seu comprometimento pode impactar a integridade e a disponibilidade de uma aplicação que faz uso desta tecnologia. Na Seção 3.2 é apresentada a arquitetura do IT-SPIRE. Por fim, na Seção 3.3 são apresentadas algumas considerações sobre o que foi explanado nesse capítulo.

3.1 COMPONENTES CRÍTICOS NO SPIRE

Os quatro componentes principais da arquitetura SPIRE mostrada na Figura 3 (página 27) são Servidor, Agente, BD e Cargas de Trabalho. Destes componentes, apenas as Cargas são supostas não confiáveis. O Servidor, os Agentes e o BD são considerados confiáveis, e são críticos tanto para a disponibilidade quanto para a integridade de aplicações que usam o SPIRE, como detalhado a seguir. O material desta seção consolida e complementa várias discussões sobre ameaças presentes, de forma dispersa, na documentação do SPIFFE (FELDMAN et al., 2020).

Um Agente é crítico para o nó em que executa. A indisponibilidade de um Agente impede qualquer Carga servida por ele de obter novos SVIDs. Além disso, o comprometimento da integridade de um Agente permite que ele forneça a uma Carga de Trabalho o SVID de qualquer Carga registrada para executar no nó, ou seja, da qual esse Agente é o *Parent ID*. De fato, o Agente é responsável por atestar uma Carga e atribuir-lhe o SVID correto; Servidor e Cargas de Trabalho não interagem diretamente, sendo o processo mediado pelo Agente. A única mitigação disponível contra um Agente comprometido é minimizar as Cargas atribuídas ao seu nó.

Um Servidor SPIRE é crítico para todo o seu domínio de confiança. A indisponibilidade do Servidor impede que novos Agentes passem pela atestação de nó e recuperem os SVIDs que lhes são atribuídos, fazendo com que não possam servir seus nós. Agentes que já estavam ativos antes da indisponibilidade do Servidor ficam impedidos de renovar SVIDs (a renovação ocorre quando o prazo de expiração do SVID cai a menos da metade), bem como emitir JWT-SVIDs para novas audiências (cada *token* JWT possui uma audiência específica, que define o destinatário pretendido do *token*, *i.e.*, quem deve processá-lo (JONES; BRADLEY; SAKIMURA, 2015)). O *cache* de SVIDs no Agente mitiga parcialmente a indisponibilidade do Servidor, sendo útil principalmente para falhas transientes na infraestrutura.

Um Servidor SPIRE comprometido pode emitir qualquer SVID no seu domínio de confiança e alterar o pacote de confiança, incluindo novas chaves de certificação. Nas arquiteturas de implantação com alta disponibilidade (SPIRE-HA e Nested SPIRE, descritas na Seção 2.3.2.3), qualquer Servidor comprometido pode gerar SVIDs arbitrários no domínio de confiança. No caso do SPIRE Federado, o Servidor de um domínio pode adulterar o pacote de confiança dos domínios federados, podendo levar as Cargas de Trabalho no seu domínio a aceitarem como válidos também SVIDs arbitrários em outros domínios.

O BD é ainda mais crítico para a arquitetura do que o Servidor SPIRE, tanto em termos de disponibilidade quanto de integridade. Ele armazena as entradas de registro referentes às Cargas de Trabalho de um domínio de confiança, e sua indisponibilidade impede o registro de novas Cargas e a emissão de SVIDs para elas, e pode impedir que Agentes obtenham e renovem SVIDs. Caso o BD seja restaurado sem voltar ao estado que tinha antes de ficar indisponível, todos os Agentes e Cargas precisarão passar novamente por atestação e emissão de SVIDs. Ele também armazena o pacote de confiança, e sua indisponibilidade pode impedir que Cargas e Agentes obtenham o pacote, dependendo de como este é distribuído. Do ponto de vista de integridade, o comprometimento do BD permite que um atacante crie entradas de registro arbitrárias, modifique as entradas existentes, manipule o pacote de confiança, e adultere informações usadas na atestação de nós. O escopo dessas manipulações de informações é restrito aos domínios de confiança e Servidores que o BD atende.

Embora seja possível adotar uma arquitetura de alta disponibilidade para o BD, isso tipicamente significa um BD replicado, que tolera falhas de parada. Tal arquitetura reduz os problemas de disponibilidade, mas ao mesmo tempo aumenta a superfície de ataque para violações de integridade. Isso ocorre porque comprometer qualquer réplica usada para tolerância a falhas de parada pode ser suficiente para manipular o conteúdo do BD.

3.2 ARQUITETURA

A tolerância a intrusões em um sistema distribuído passa por eliminar ou reduzir pontos únicos de falha na arquitetura (VERÍSSIMO; NEVES; CORREIA, 2003). Como discutido na Seção 3.1, existem três componentes críticos na arquitetura do SPIRE: Agente, Servidor e BD. Tipicamente, um domínio de confiança possui um Servidor (com alternativas arquiteturais para alta disponibilidade), um BD e vários Agentes. Violações de disponibilidade e integridade do Servidor e do BD podem afetar um domínio de confiança inteiro, enquanto que problemas com o Agente têm efeito limitado ao nó em que esse Agente executa (que pode abrigar várias Cargas de Trabalho). Portanto, Servidor e BD são mais críticos para a arquitetura do que o Agente, ainda que o comprometimento de vários Agentes possa afetar um número significativo de Cargas de Trabalho. Ademais, a premissa de que cada nó possui um Agente dificulta a adaptação da arquitetura para usar uma variante distribuída de Agente, tornando esse componente mais complicado de substituir por uma versão TI.

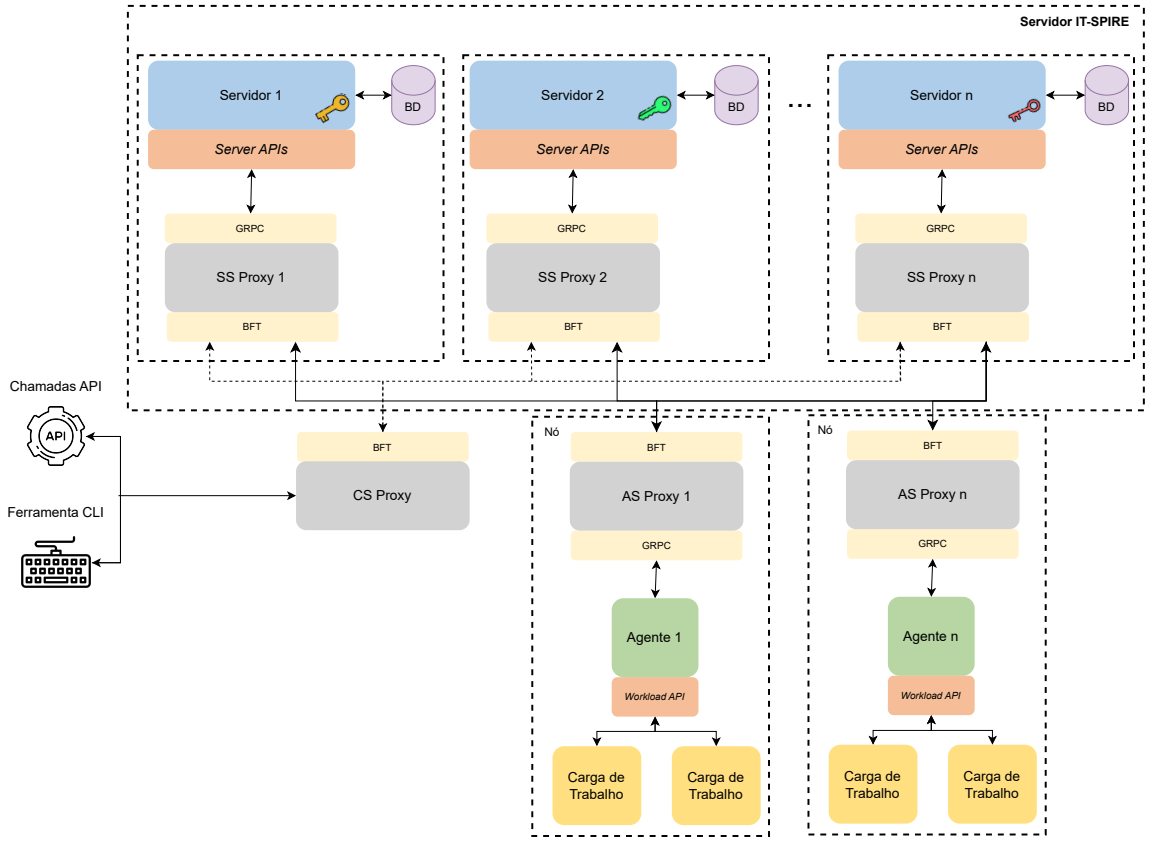
Os dois componentes mais críticos, Servidor e BD, oferecem riscos parecidos considerando violações tanto de disponibilidade quanto de integridade. Uma versão TI do Servidor tem como requisito básico impedir que um Servidor comprometido gere SVIDs arbitrários dentro do domínio de confiança.

O BD armazena as entradas de registro usadas para identificar Cargas de Trabalho, os SVIDs emitidos pelo Servidor e o pacote de confiança do domínio. Como o BD é desacoplado do Servidor, seria possível substituí-lo por um serviço tolerante a intrusões. Uma dificuldade aqui é

que, embora o Servidor SPIRE suporte bancos de dados variados por meio de *plugins*, os *plugins* disponíveis são todos compatíveis com SQL, o que exigiria um serviço de armazenamento tolerante a intrusões com interface SQL, tornando a adaptação mais complicada. Uma outra questão é que um Servidor bizantino pode simplesmente ignorar o BD tolerante a intrusões e emitir os SVIDs ou gerar o pacote de confiança que quiser (ele pode até usar um BD próprio, como um SQLite (SQLITE, 2024)). Ou seja, tornar o BD mais seguro resolve os problemas advindos de um BD comprometido, mas não resolve os problemas de um Servidor comprometido.

Sendo assim, o caminho mais promissor para reduzir os componentes críticos na arquitetura SPIRE, tornando-a mais resiliente, é desenvolver versões tolerantes a intrusões para o Servidor, o BD e o Agente (nessa ordem) que venham a substituir os componentes atuais. A sequência proposta prioriza os componentes cuja substituição oferece o melhor custo-benefício. Portanto, esta dissertação de mestrado se concentra na proposição do IT-SPIRE, um Servidor SPIRE tolerante a intrusões. A arquitetura proposta está ilustrada na Figura 8.

Figura 8 – Arquitetura IT-SPIRE



Fonte: Elaborado pela autora (2024).

Um domínio de confiança possui um conjunto de n Servidores que usam replicação máquina de estados. Cada réplica tem o seu próprio BD local, que é independente dos demais; assim, do ponto de vista da arquitetura, Servidor e BD formam um único componente, sujeito a

falhas bizantinas. Dessa forma, se um BD for comprometido, a respectiva réplica do Servidor será vista como incorreta. Consequentemente, a arquitetura do IT-SPIRE também mitiga os problemas de disponibilidade e integridade do Banco de Dados. Para tolerar f Servidores comprometidos, são necessárias $n \geq 3f + 1$ réplicas.

A comunicação entre Agentes e Servidores é realizada por meio de dois *proxies*. Estes foram criados com o intuito de minimizar as modificações no código existente. O *Agent-Side Proxy* (AS Proxy) tem como objetivo intermediar a comunicação entre Agente e Servidores. Este *proxy* é implementado do lado do Agente, possibilitando a não modificação do código deste. Já o *Server-Side Proxy* (SS Proxy) é implementado do lado do Servidor para intermediar a comunicação entre Servidor e demais componentes da arquitetura.

O AS Proxy possui uma interface igual à do Servidor SPIRE, fazendo com que o Agente ache que está se comunicando com o Servidor SPIRE final. Entretanto, ao receber a requisição do Agente via gRPC, o AS Proxy implementa o lado do cliente BFT e envia a requisição serializada para o SS Proxy. Já o SS Proxy implementa o lado do servidor BFT, ou seja, são criadas $n \geq 3f + 1$ réplicas do SS Proxy. Este recebe a requisição do cliente (AS Proxy) e é responsável por fazer as chamadas gRPC para o Servidor SPIRE. O AS Proxy irá receber uma resposta de cada réplica; estas respostas são idênticas, com exceção das respostas que necessitam da assinatura dos servidores. Nestes casos as assinaturas diferem entre si, mas o restante dos dados são iguais.

O acesso à *Server API* responsável pelo gerenciamento de entradas de registro também é feito via BFT. O *Client-Side Proxy* (CS Proxy) serializa a requisição com as entradas de registro e envia para o SS Proxy. Este deserializa a requisição e realiza então a chamada gRPC para o Servidor SPIRE. Dessa forma é possível garantir que todas as réplicas tenham uma visão consistente das entradas de registro do domínio.

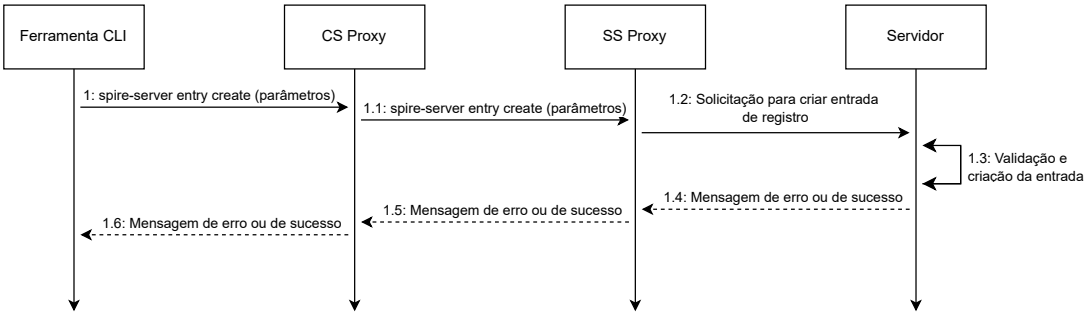
O funcionamento interno do Agente, o seu papel na arquitetura e a interação entre Cargas de Trabalho e Agente permanecem inalterados. A seguir é descrito o funcionamento das principais operações na arquitetura proposta, sendo organizadas de acordo com a ordem de execução. Ou seja, primeiramente o Servidor gerencia o pacote de confiança, em seguida acontece o gerenciamento de entradas de registro e a geração do *join token* (método escolhido para a autenticação entre Agente e Servidor). Da parte do Agente, após o recebimento do *join token*, o mesmo solicita a atestação do seu respectivo nó, para que em seguida possa realizar o processo de sincronização para adquirir as entradas de registro para as quais o Agente está autorizado, bem como solicitar ao Servidor a emissão dos SVIDs das mesmas. Por fim, após o recebimento dos SVIDs das entradas, o Agente pode realizar a atestação das Cargas de Trabalho.

Gerência do pacote de confiança: cada um dos n Servidores possui seu próprio par de chaves pública e privada. Portanto, há n pares de chaves, que são gerenciadas da forma habitual. A única mudança é que o pacote de confiança do domínio passa a ter as chaves públicas de todos os n Servidores. Após receber $f + 1$ pacotes idênticos, o AS Proxy encaminhará para o Agente o

último pacote recebido.

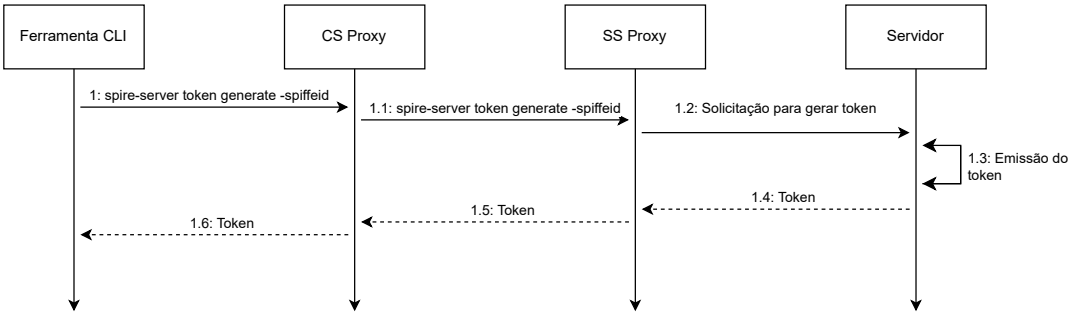
Gerenciamento de entradas de registro e geração do *join token*: as operações referentes a criação, leitura, alteração e remoção de entradas de registro são realizadas usando BFT. Para as operações que modificam a base de dados de entradas de registro (criação, alteração e remoção), isso garante que todos os servidores (corretos) tenham uma visão consistente da base. Para operações de consulta, isso garante que uma minoria de até f servidores comprometidos não possa falsear as informações contidas na base. Conforme mencionado na Seção 2.3.2.1, as entradas de registro podem ser manipuladas usando uma interface de linha de comando. A Figura 9 apresenta um exemplo dos fluxos entre a Ferramenta CLI, o CS Proxy, o SS Proxy e o Servidor para a criação de uma entrada de registro. Os mesmos fluxos são aplicados para operações de leitura, alteração e remoção de entradas, bem como para a geração de *join tokens*. A Figura 10 apresenta os fluxos para a geração de um *join token*.

Figura 9 – Fluxo de criação de entradas de registro



Fonte: Elaborado pela autora (2024).

Figura 10 – Fluxo de emissão do *join token*



Fonte: Elaborado pela autora (2024).

Atestação de Nó: a Figura 11 apresenta os fluxos para a atestação de nó. Um Agente envia

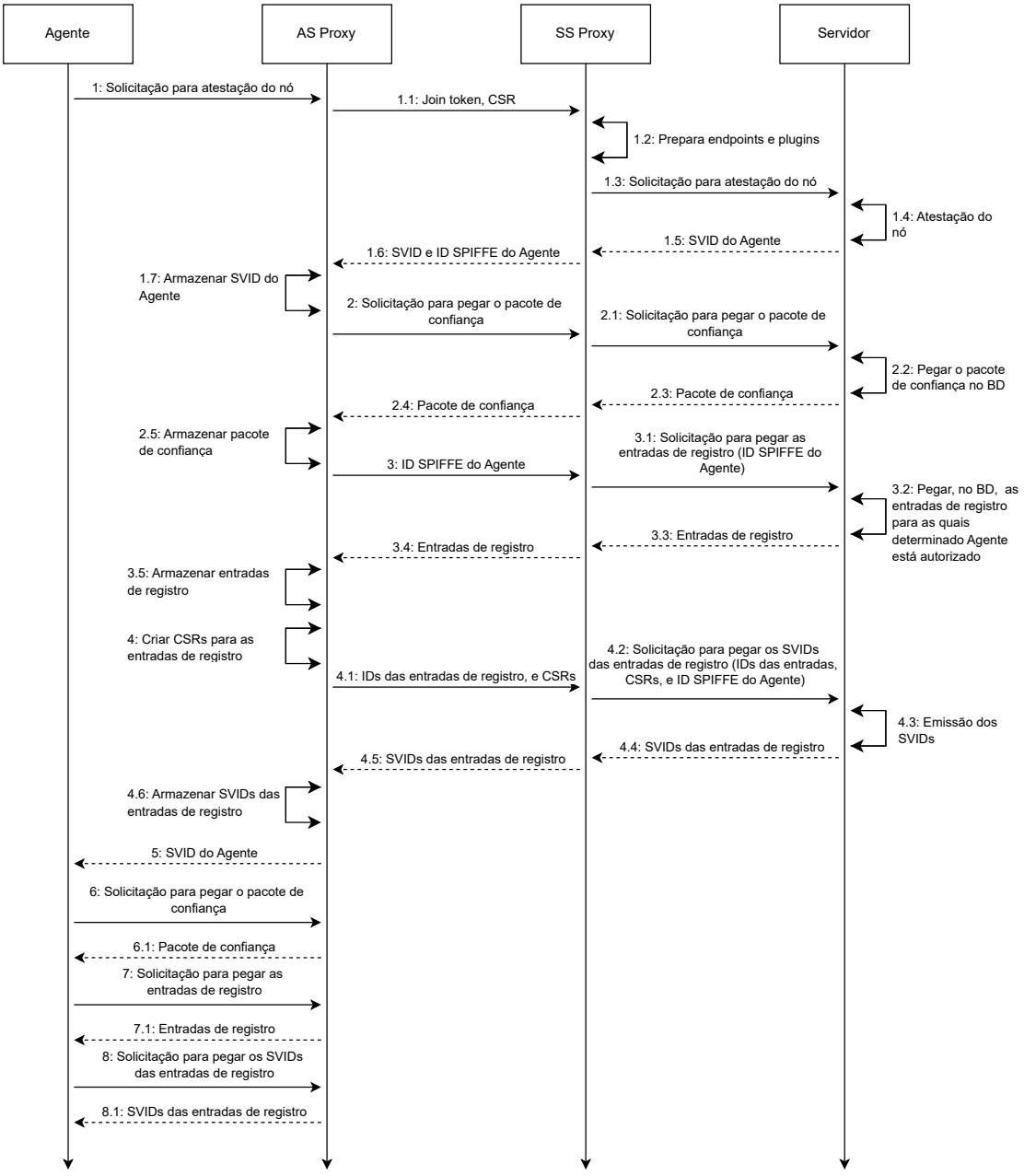
suas evidências de atestação para todos os n Servidores por meio dos Proxies AS e SS, os quais usam BFT (fluxos 1 até 1.3). Com isso, todos os Servidores recebem as mesmas evidências. Cada Servidor usa o seu BD e interage com os atestadores de nó para atestar o nó do Agente (fluxo 1.4). Se a atestação for bem sucedida, o Servidor envia o SVID do Agente ao SS Proxy (fluxo 1.5), o qual o envia para o AS Proxy (fluxo 1.6). Ao receber pelo menos $f+1$ SVIDs contendo o mesmo ID SPIFFE, o AS Proxy armazena localmente o último SVID recebido dos SS Proxies (fluxo 1.7). Em seguida, o AS Proxy envia o ID SPIFFE do domínio de confiança para o SS Proxy (fluxo 2). Este o envia para o Servidor (fluxo 2.1), o qual retorna o respectivo pacote de confiança (fluxo 2.3). O AS Proxy então armazena localmente o pacote de confiança recebido (fluxo 2.5), e em seguida envia o ID SPIFFE do Agente para adquirir as entradas de registro as quais o Agente está autorizado (fluxo 3). O SS Proxy encaminha este ID SPIFFE do Agente ao Servidor (fluxo 3.1), o qual retorna as devidas entradas de registro (fluxo 3.3). Estas são armazenadas pelo AS Proxy (fluxo 3.5). Para cada entrada de registro, o AS Proxy cria um CSR (*Certificate Signing Request*) (fluxo 4), o qual é enviado, juntamente com o ID das entradas, para o SS Proxy (fluxo 4.1), com o intuito de adquirir os SVIDs das respectivas entradas de registro. O SS Proxy encaminha essa solicitação para o Servidor (fluxo 4.2), e este emite os SVIDs necessários (fluxo 4.3), os quais são armazenados pelo AS Proxy (fluxo 4.6). Apenas após a coleta dessas informações (pacote de confiança, entradas de registro, e SVIDs das entradas de registro) é que o SVID adquirido na atestação do nó é enviado para o Agente (fluxo 5). Para encerrar o processo de Atestação do nó, o Agente envia a solicitação para pegar o pacote de confiança (fluxo 6). O AS Proxy retorna o pacote de confiança armazenado anteriormente (fluxo 6.1).

Processo de Sincronização: conforme observado na Figura 11, após o Agente receber o pacote de confiança, este irá solicitar as entradas de registro (fluxo 7). O AS Proxy retorna as entradas armazenadas anteriormente (fluxo 7.1). Por fim, o Agente solicita os SVIDs das entradas de registro (fluxo 8). As duas últimas solicitações fazem parte do gerenciamento do *cache* do Agente, *i.e.*, essas requisições são executadas periodicamente em um intervalo de tempo predefinido (5 segundos por padrão).

Emissão de SVIDs: cada Servidor emite um SVID com base nas entradas de registro existentes no seu BD. O SVID é assinado com a chave privada do Servidor.

Atestação de Carga de Trabalho: o Agente permanece responsável por atestar as suas Cargas de Trabalho. Cada SVID associa um ID SPIFFE a uma Carga, representada por seus seletores. A alteração é que, com n Servidores, na atestação do nó, o AS Proxy recebe n SVIDs para cada Carga. O mesmo acontece caso o Agente necessite solicitar a emissão de um novo SVID; essa solicitação é realizada usando BFT, garantindo *(i)* que todos os n servidores recebam os mesmos seletores e *(ii)* que todos os servidores corretos (e consistentes) emitam SVIDs com o mesmo ID SPIFFE. Para uma dada Carga, ao receber pelo menos $f+1$ SVIDs contendo o

Figura 11 – Fluxos de atestação de nó e sincronização



Fonte: Elaborado pela autora (2024).

mesmo ID SPIFFE, o AS Proxy fornece ao Agente o último SVID recebido dos SS Proxies. Assim, do ponto de vista das Cargas de Trabalho e do Agente, a arquitetura IT-SPIRE é indistinta da arquitetura SPIRE não tolerante a intrusões; os SVIDs recebidos (sejam certificados X.509 ou *tokens* JWT) são idênticos aos da arquitetura convencional, e podem continuar sendo usados da mesma maneira.

Arquiteturas de Implantação: considerando as arquiteturas de implantação apresentadas na Seção 2.3.2.3, a arquitetura IT-SPIRE pode ser considerada uma evolução da arquitetura de alta disponibilidade SPIRE-HA, estando restrita a um domínio de confiança. Por ora, não são contempladas as arquiteturas Nested SPIRE ou SPIRE Federado.

3.3 CONSIDERAÇÕES PARCIAIS

Primeiramente, este capítulo identificou os componentes críticos do SPIRE, e o caminho mais promissor para reduzir esses componentes, tornando a arquitetura do SPIRE mais resiliente. Dentre os quatro componentes principais do SPIRE (Agente, BD, Carga de Trabalho, e Servidor), a substituição do Servidor por um Servidor tolerante a intrusões apresentou o melhor custo-benefício. Portanto, o objetivo desta dissertação é propor o IT-SPIRE, um Servidor SPIRE tolerante a intrusões. A arquitetura do IT-SPIRE foi descrita neste capítulo.

O próximo capítulo apresenta a implementação de um protótipo do IT-SPIRE, bem como realiza uma avaliação experimental do mesmo. Esta avaliação é realizada a partir de dados coletados pelo Agente SPIRE durante o processo de inicialização e de sincronização em ambas as arquiteturas (IT-SPIRE e SPIRE).

4 IMPLEMENTAÇÃO E AVALIAÇÃO EXPERIMENTAL

O Servidor IT-SPIRE foi implantado na rede da Universidade do Estado de Santa Catarina (UDESC). Este capítulo apresenta a implementação do protótipo do IT-SPIRE, e realiza uma avaliação experimental, analisando os dados coletados pelo Agente SPIRE durante a atestação do nó e a atualização do SVID do Agente, bem como o tempo de sincronização. A Seção 4.1 apresenta a implementação do protótipo. A Seção 4.2 introduz as definições operacionais da implantação do protótipo do IT-SPIRE. A Seção 4.3 discute a avaliação experimental do protótipo. Por fim, a Seção 4.4 apresenta algumas considerações sobre o que foi explanado nesse capítulo.

4.1 IMPLEMENTAÇÃO

Um dos objetivos desta dissertação é construir um protótipo da arquitetura do IT-SPIRE a fim de demonstrar a viabilidade da proposta e obter uma estimativa inicial do seu impacto em termos de desempenho. Para a implementação de replicação máquina de estados foi usado o BFT-SMaRt, uma biblioteca de código aberto que implementa RME tolerante a faltas bizantinas ou de parada, com uma semântica de falhas configurável (BESSANI; SOUSA; ALCHIERI, 2014). Além disso, esta biblioteca fornece protocolos para reconfiguração e gerenciamento de estado (*checkpoints*, transferência e atualização de estado). Embora o BFT-SMaRt seja implementado em Java, existe uma camada intermediária que permite que a biblioteca seja usada com outras linguagens de programação (C, C++, Go e Python) (COSTA; ALCHIERI, 2018). Como o SPIRE é implementado em Go, este trabalho faz uso da interface BFT-SMaRt Go para implementar o CS Proxy, o AS Proxy e o SS Proxy (Figura 8). Embora o projeto usando *proxies* vise isolar grande parte do código SPIRE dos mecanismos de replicação máquina de estados, foram necessárias algumas pequenas alterações no Servidor, conforme descrito abaixo. Neste ponto, uma implementação completa não era um objetivo, então foi implementado/adaptado apenas o código necessário para teste e avaliação.

Gerenciamento de entradas de registro: no Servidor SPIRE, o ID das entradas é gerado de forma aleatória. Quando o Servidor é replicado, isso compromete o determinismo de réplica (Seção 2.2): uma entrada recebe um ID diferente em cada réplica, o que significa que o BFT-SMaRt tratará respostas idênticas (a não ser pelo ID) como diferentes e não conseguirá chegar a um consenso. Para contornar isso, o Servidor IT-SPIRE gera o ID das entradas com base no ID SPIFFE desta e em um *namespace* predefinido. Portanto, todos os Servidores corretos geram o mesmo ID para a mesma entrada, e entradas de Servidores diferentes serão consideradas iguais pelo comparador BFT-SMaRt. Cabe observar que a documentação do SPIFFE não fornece nenhuma justificativa de segurança para a aleatoriedade na geração de IDs de entrada.

Atestação de Nó: para simplificar, o protótipo usa um *join token* para atestar o nó do Agente (SPIFFE, 2019a), o que é suficiente para fins de teste. No SPIRE, o *token* é gerado aleatoriamente,

assim como os IDs das entradas de registro. Portanto, devido à replicação do Servidor, no protótipo IT-SPIRE a geração do *token* é baseada no ID SPIFFE fornecido, e em um *namespace* predefinido. Consequentemente, todos os n Servidores geram o mesmo *token* para o mesmo conjunto de entrada (*namespace* e ID SPIFFE).

A ideia inicial da integração do BFT-SMaRt com o SPIRE está ilustrada na Figura 12. Porém, devido a problemas com a integração do BFT-SMaRt que causaram alguns travamentos no protótipo, em vez do AS Proxy enviar as respostas recebidas do SS Proxy diretamente para o Agente (fluxos 1.7, 2.6, 3.6, e 4.6), o AS Proxy deve armazenar estes resultados localmente, conforme observado nos fluxos 1.7, 2.5, 3.5 e 4.6 da Figura 11. Somente depois de coletar o SVID do Agente, o pacote de confiança, as entradas de registro, e os SVIDs das entradas, o AS Proxy envia o SVID para o respectivo Agente.

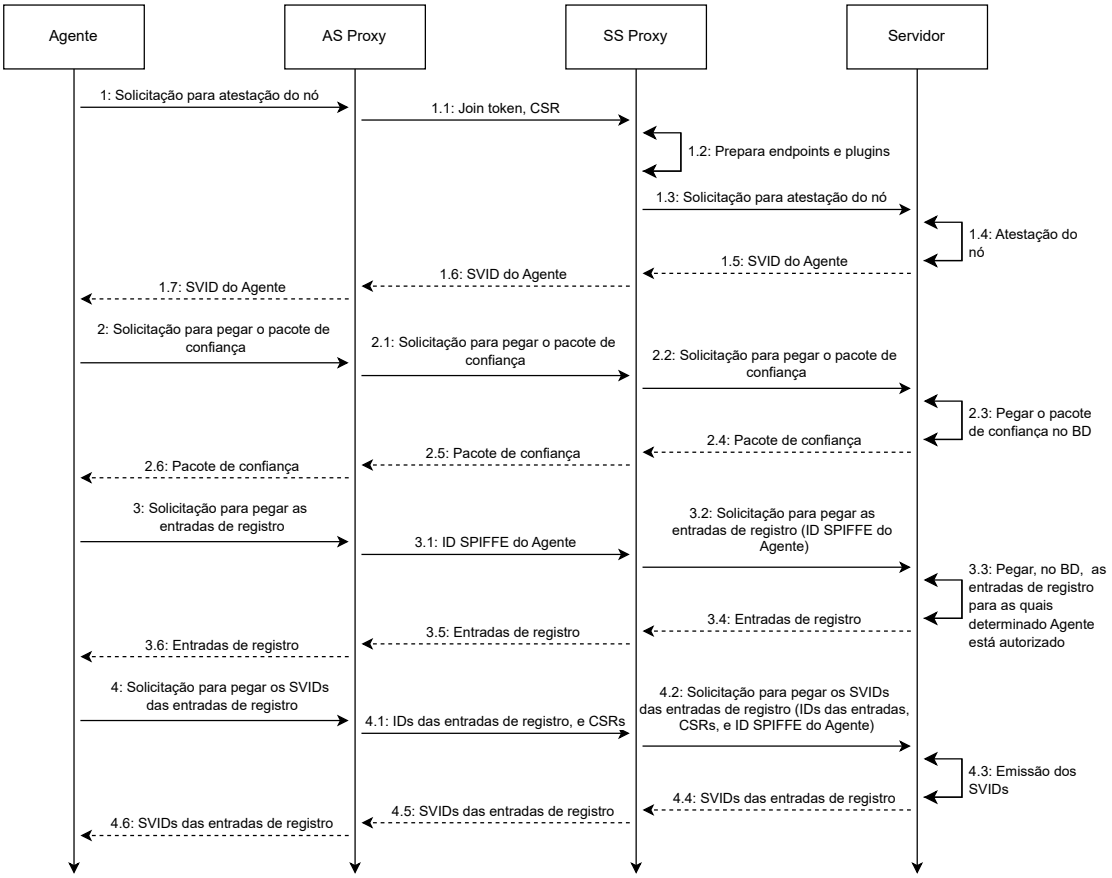
Com base nos experimentos de Costa e Alchieri (2018), o CS Proxy e o AS Proxy foram configurados para aguardar $2f + 1$ respostas iguais. Conforme mencionado na Seção 3.2, estas respostas são idênticas, com exceção das respostas que necessitam da assinatura dos servidores. Dessa forma o comparador do BFT-SMaRt deve comparar as respostas *byte* a *byte*, ignorando as diferenças de assinatura.

Processo de Sincronização: o processo de sincronização do SPIRE acontece em *loop*, entretanto, devido a problemas encontrados na integração com o BFT-SMaRt, até o momento foi possível implementar apenas a execução de uma iteração dessa sincronização.

Na arquitetura do SPIRE, após a atestação do nó, o Agente usa seu respectivo SVID para se autenticar ao Servidor via mTLS. Já no protótipo do IT-SPIRE, para simplificação, a comunicação entre o SS Proxy e o Servidor é realizada via *socket* local. Consequentemente, foi necessário alterar as APIs *entry* e *svid* do Servidor para acrescentar o ID SPIFFE do Agente na chamada gRPC para pegar as entradas de registro (*GetAuthorizedEntriesRequest*) e os SVIDs das mesmas (*BatchNewX509SVIDRequest*). Essas modificações podem ser observadas no GitHub da autora (GITHUB, 2024).

Formato dos SVIDs: apesar do SPIRE suportar SVIDs no formato JWT, o protótipo do IT-SPIRE concentra-se no formato X.509, o que é suficiente para a avaliação da proposta.

Figura 12 – Fluxos de atestação de nó e sincronização desejados



Fonte: Elaborado pela autora (2024).

4.2 AMBIENTE EXPERIMENTAL

O ambiente de software utilizado foi o sistema operacional Ubuntu 22.04 LTS, JVM Oracle JDK 1.8.0 131 (Java), g++ 11.4.0 (C++), gcc 11.4.0 (C), Go 1.21.5, e Docker 24.0.5. O BFT-SMaRt foi configurado com $n = 4$ servidores para tolerar até uma falha bizantina. Neste trabalho, os clientes BFT-SMaRt foram configurados para aguardar $2f + 1$ respostas iguais. Além disso, foram utilizadas cinco máquinas (ens1, ens2, ens3, ens4, ens5) com as mesmas configurações de software. A configuração de hardware é heterogênea, como apresentado na Tabela 4. Cada Servidor foi implantado em uma máquina diferente, juntamente com seu respectivo SS Proxy. Ou seja, a primeira réplica foi implantada na máquina ens1, a segunda na ens2, e assim por diante. Já o CS Proxy, o AS Proxy e o Agente foram implantados na mesma máquina (ens5). Essa estrutura pode ser observada na Figura 13.

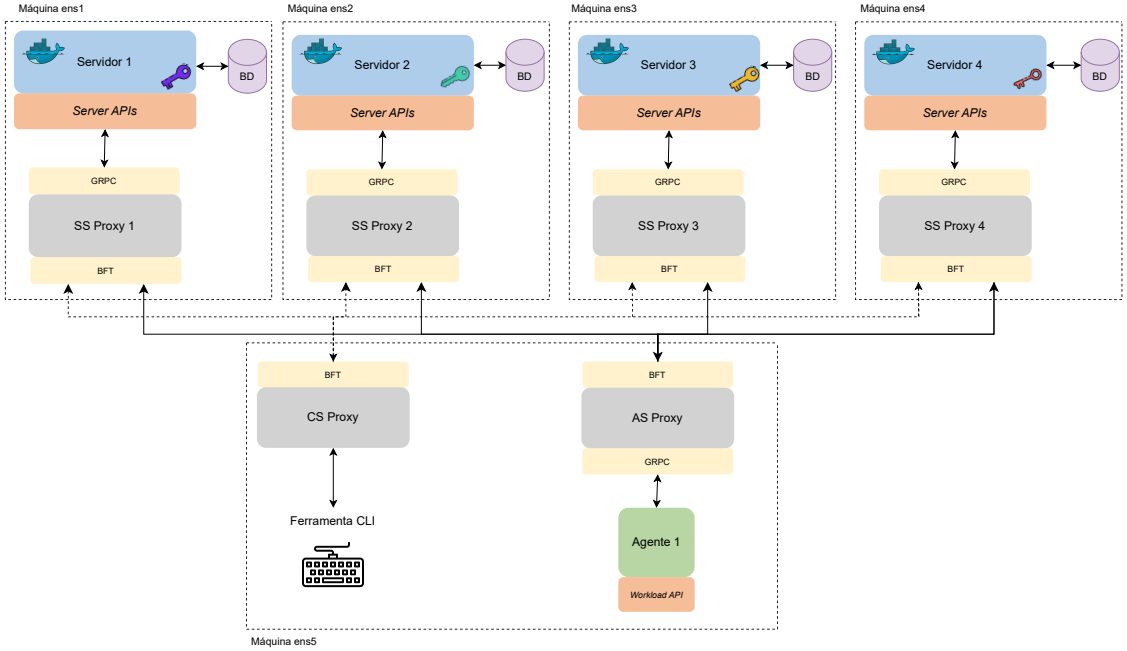
Para fins de comparação, o SPIRE foi implantado em duas máquinas, sendo o Servidor na ens4, e o Agente na ens5. Vale ressaltar que, a partir do código fonte do SPIRE, foi gerada uma

Tabela 4 – Configuração do ambiente de teste

	Processador			Memória (GB)	Disco			Rede	
Máquina	Modelo	Quant. núcleos	Clock (GHz)		Modelo	Capacidade (GB)	HD x SSD	Modelo	Velocidade (Mbps)
ens1	Intel(R) Core(TM) i5-8500	6	3.00	16	KINGSTON SA400S3	447.1	SSD	Ethernet	1000
ens2	Intel(R) Core(TM) i5-8500	6	3.00	16	KINGSTON SA400S3	447.1	SSD	Ethernet	1000
ens3	Intel(R) Core(TM) i5-8500	6	3.00	16	KINGSTON SA400S3	447.1	SSD	Ethernet	1000
ens4	Intel(R) Core(TM) i7-4770	8	3.40	16	WDC WD5000AAKX-0	465.8	HD	Ethernet	1000
ens5	Intel(R) Core(TM) i7-4770	8	3.40	16	WDC WD5000AAKX-0	465.8	HD	Ethernet	1000

Fonte: Elaborado pela autora (2024).

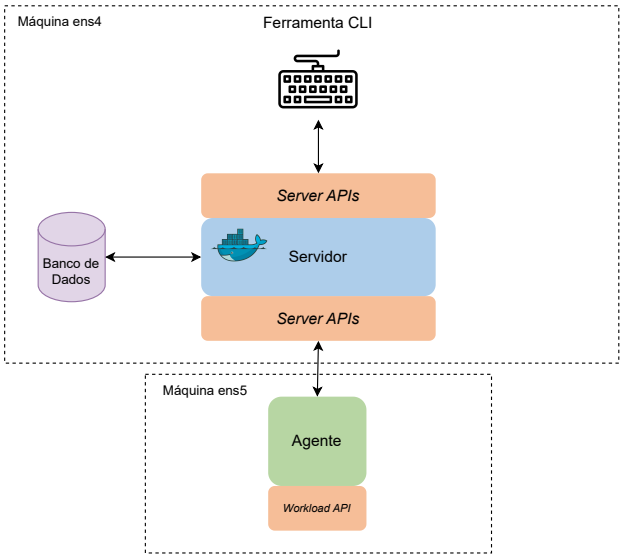
Figura 13 – Arquitetura de implantação do IT-SPIRE



Fonte: Elaborado pela autora (2024).

imagem de contêiner para o Servidor. A estrutura de implantação do SPIRE pode ser observada na Figura 14.

Figura 14 – Arquitetura de implantação do SPIRE



Fonte: Elaborado pela autora (2024).

4.3 AVALIAÇÃO EXPERIMENTAL

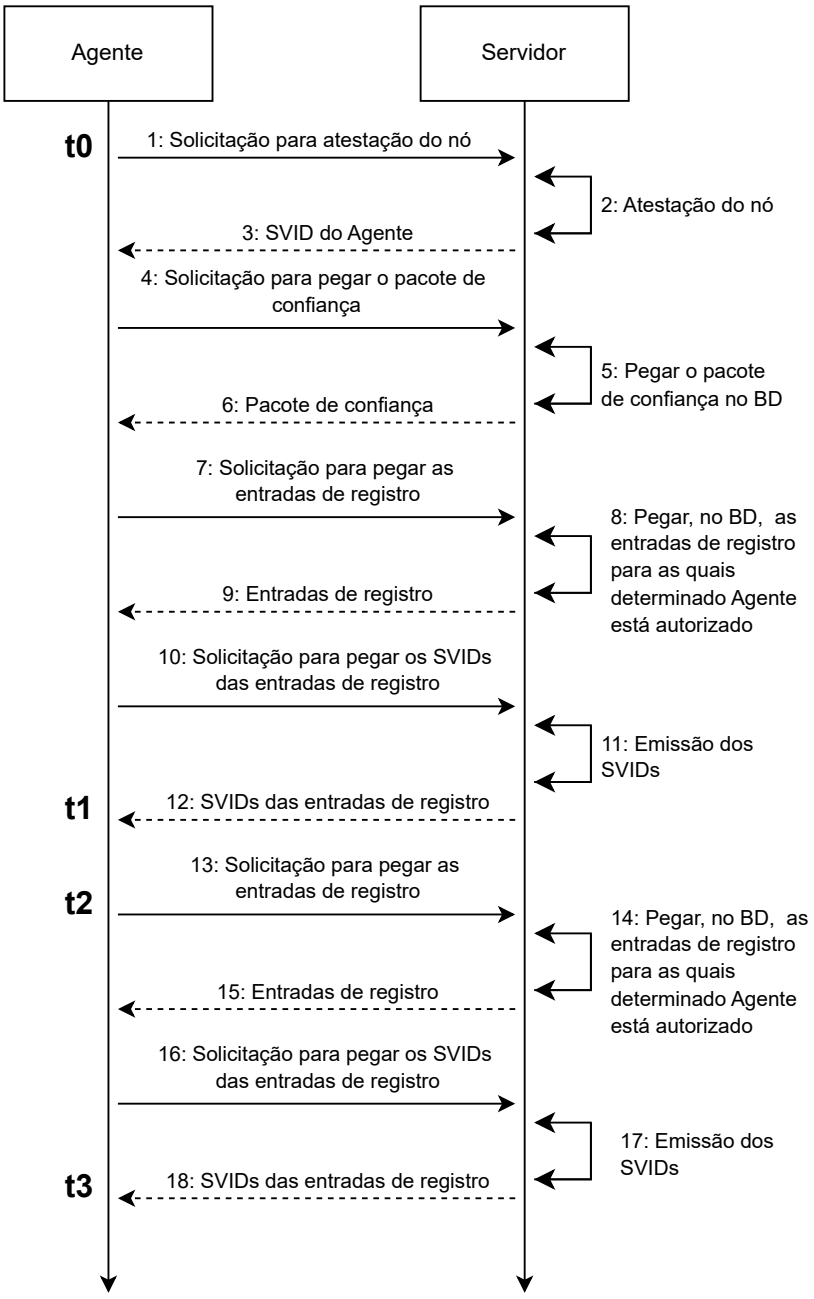
O protótipo foi submetido a testes funcionais, cujo sucesso mostrou que a proposta é factível. Além dos testes funcionais, foram realizados alguns experimentos com vistas a obter estimativas iniciais do impacto da proposta sob a ótica do desempenho. Foi considerado apenas um cenário livre de faltas, uma vez que a tolerância a faltas e intrusões é encapsulada pela biblioteca BFT-SMaRt, cujo desempenho em cenários com faltas e reconfigurações foi avaliado extensivamente por Bessani, Sousa e Alchieri (2014).

Os experimentos permitiram avaliar o tempo de inicialização das arquiteturas IT-SPIRE e SPIRE, *i.e.*, o tempo que o Agente SPIRE leva para inicializar e ficar pronto para atender as Cargas de Trabalho (Seção 4.3.1). Também foi avaliado o tempo de sincronização, *i.e.*, o período de sincronização no qual o Agente solicita as entradas de registro para as quais está autorizado, bem como os SVIDs para as mesmas (Seção 4.3.2). Cada experimento foi executado 40 vezes.

4.3.1 Tempo de Inicialização

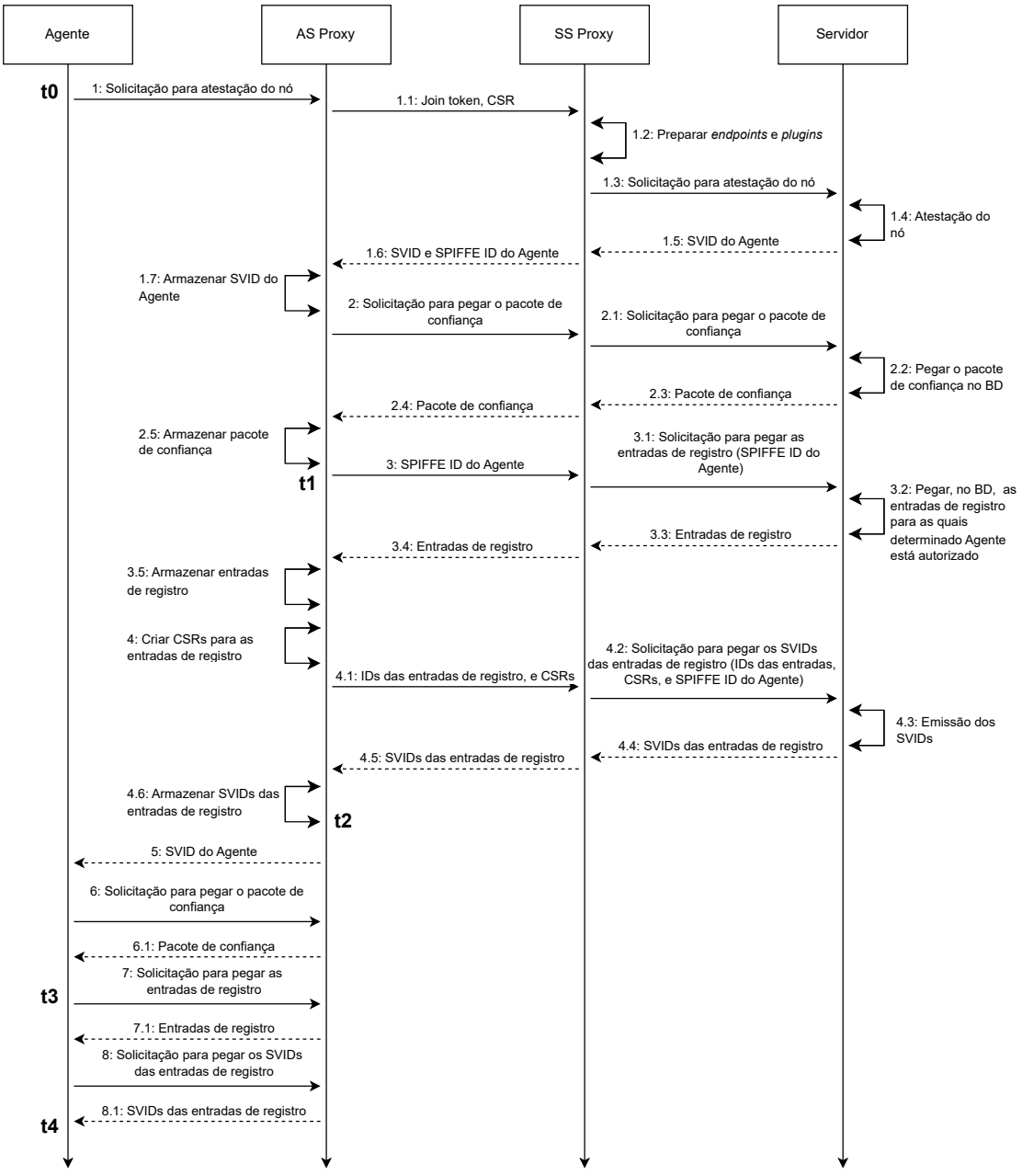
Durante a atestação do nó, usando o método *Join Token*, é criada uma entrada de registro para o Agente. Esta é marcada como obsoleta para que, durante o processo de sincronização, o SVID do Agente seja atualizado. Portanto, a medição de inicialização para o SPIRE consiste no tempo entre a solicitação do Agente para a atestação do nó (t0 da Figura 15, e t0 da Figura 16 para o IT-SPIRE), até o primeiro *loop* de sincronização para atualizar o SVID do Agente (t1 da Figura 15 para o SPIRE, e t4 da Figura 16 para o IT-SPIRE).

Figura 15 – Medição dos tempos no SPIRE



Fonte: Elaborado pela autora (2024).

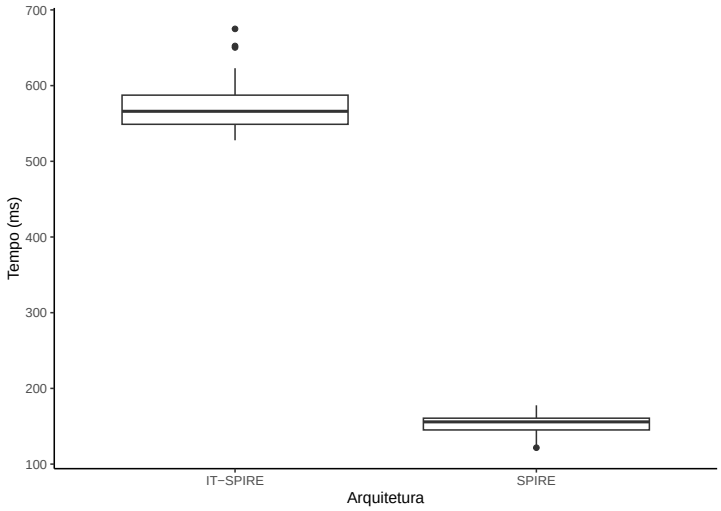
Figura 16 – Medição dos tempos no IT-SPIRE



Fonte: Elaborado pela autora (2024).

Tanto o SPIRE quanto o IT-SPIRE foram executados 40 vezes para obter o respectivo tempo de inicialização. A distribuição dos valores obtidos pode ser observada no *boxplot* da Figura 17. O tempo de inicialização para o SPIRE levou, em média, 152,5 ms, com desvio padrão de 13,5 ms. Já o IT-SPIRE teve uma média de 573 ms, com desvio padrão de 34,8 ms. A Tabela 5 fornece um resumo estatístico com médias, medianas, desvios padrão, intervalos de confiança de 95%, e coeficientes de variação (CVs) para os dados medidos.

Figura 17 – Boxplots das medições de tempo de inicialização



Fonte: Elaborado pela autora (2024).

Tabela 5 – Estatísticas para as medições de tempo de inicialização

	Média (ms)	Mediana (ms)	Desvio padrão (ms)	IC 95% (ms)	CV
SPIRE	152,5	155,8	13,5	[148,3; 156,7]	8,9%
IT-SPIRE	573,0	566,1	34,8	[562,2; 583,8]	6,1%

Fonte: Elaborado pela autora (2024).

Observando o *boxplot* da Figura 17, nota-se que o tempo médio de inicialização aumentou de 152,5 ms (SPIRE) para 573 ms (IT-SPIRE). Essa sobrecarga de tempo de 275,74% é introduzida por conta dos *proxies* do lado do Agente e do Servidor e pela replicação dos servidores, que adiciona trocas de mensagens e processamento nos nós. Embora o aumento relativo não seja pequeno, em termos absolutos o tempo de inicialização do IT-SPIRE pode ser considerado aceitável, tendo em vista a segurança adicional oferecida e o fato de que é um impacto restrito à inicialização do Agente (ou seja, não recorrente).

4.3.2 Tempo de Sincronização

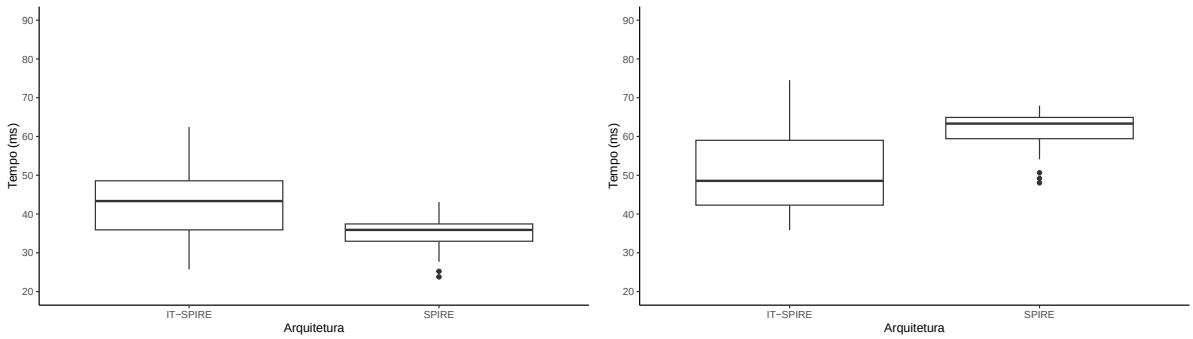
No SPIRE, a medição do tempo de sincronização consiste no *loop* de sincronização para pegar as entradas de registro para as quais o Agente SPIRE está autorizado, e os SVIDs

das mesmas (t2 até t3 da Figura 15). Já no IT-SPIRE, devido a problemas na integração com o BFT-SMaRt, a medição do tempo de sincronização consiste na soma dos intervalos [t1;t2] e [t3;t4] da Figura 16. Neste trabalho foram testados os tempos para 10 e 20 entradas. No IT-SPIRE serão obtidas 11 e 21 entradas respectivamente, porque o primeiro *loop* de sincronização (o qual é considerado no tempo de inicialização da Seção 4.3.1) estará incluso e irá acrescentar a entrada de registro do Agente para que o SVID do mesmo seja atualizado. As imprecisões introduzidas pelas medições indiretas e pelas entradas de registro extras são consideradas sem importância, uma vez que o objetivo aqui é ter uma ideia geral do impacto no desempenho do IT-SPIRE.

A distribuição dos valores obtidos para as arquiteturas SPIRE e IT-SPIRE para 10 entradas pode ser observada no *boxplot* da Figura 18a. Já para 20 entradas, a distribuição dos valores é apresentada no *boxplot* da Figura 18b. A Tabela 6 fornece um resumo estatístico do tempo de sincronização no SPIRE e IT-SPIRE. Aqui esperava-se tempos de sincronização maiores no IT-SPIRE devido à replicação do Servidor e à adição do SS Proxy e do AS Proxy. Para 10 entradas a expectativa se confirmou: o tempo médio de sincronização no IT-SPIRE teve um aumento de 8,34 ms (23,98%) em relação ao SPIRE. Para 20 entradas, ao contrário, o tempo médio de sincronização para o IT-SPIRE foi 12,37 ms (17,50%) menor do que para o SPIRE. Os testes estatísticos (utilizando o teste *t* de Welch para duas amostras) mostram que todas as diferenças entre as médias são estatisticamente significativas, ou seja, as conclusões se mantêm mesmo quando se leva em consideração a variabilidade. Por enquanto não foi possível encontrar uma explicação para esta descoberta, mas pretende-se investigá-la mais aprofundadamente à medida que a implementação do IT-SPIRE evolui.

Figura 18 – Boxplots das medições de tempo de sincronização

(a) 10 entradas (b) 20 entradas



Fonte: Elaborado pela autora (2024).

Também esperava-se que os tempos de sincronização aumentassem com o número de entradas de registro, devido ao aumento do tempo de processamento no Servidor e ao tamanho maior das mensagens. Esta expectativa confirmou-se tanto para o SPIRE como para o IT-SPIRE: no SPIRE houve um aumento de 77,60% no tempo entre 10 e 20 entradas, enquanto no IT-SPIRE o aumento foi de 18,18%. As diferenças entre as médias são estatisticamente significativas tanto para o SPIRE como para o IT-SPIRE.

Tabela 6 – Estatísticas para as medições de tempo de sincronização

		Média (ms)	Mediana (ms)	Desvio padrão (ms)	IC 95% (ms)	CV
10 entradas	SPIRE	34,78	35,91	3,97	[33,55; 36,01]	11,41%
	IT-SPIRE	43,12	43,35	9,57	[40,15; 46,08]	22,19%
20 entradas	SPIRE	61,77	63,33	5,06	[60,20; 63,34]	8,19%
	IT-SPIRE	50,96	48,56	10,71	[47,64; 54,28]	21,01%

Fonte: Elaborado pela autora (2024).

A variabilidade dos tempos de sincronização não é pequena, com um CV mínimo de 8,2% (quanto maior o CV, maior a variabilidade dos dados). A variabilidade no protótipo IT-SPIRE foi de 21,0–22,2%, o que pode ser atribuído a uma combinação de três fatores: a variabilidade nos tempos do SPIRE, a variabilidade no BFT-SMaRt e a imaturidade do próprio protótipo, desenvolvido como prova de conceito. Em todos os experimentos com SPIRE (considerando as Tabelas 5 e 6), o menor CV foi de 8,2%; é razoável supor que um protótipo construído com base neste código apresente uma variabilidade ainda maior. Em relação ao BFT-SMaRt, a avaliação experimental de Costa e Alchieri (2018) mostra CVs na ordem de 10–15% para a latência de transmissão de mensagens, mas não há medições para a versão Go (que foi usada no protótipo); portanto, embora espera-se que esta variabilidade seja comparável, há alguma incerteza aqui. A implicação destas observações é que é difícil isolar a contribuição de cada fator para a variabilidade nos tempos do IT-SPIRE sem medições adicionais.

No geral, e tendo em conta a variabilidade dos resultados, pode-se concluir que as diferenças nos tempos de sincronização entre IT-SPIRE e SPIRE são pequenas e podem ser tratadas como aproximadamente comparáveis. Conclui-se também que este tempo de sincronização constitui uma sobrecarga razoável para o intervalo de sincronização padrão de 5 segundos.

4.4 CONSIDERAÇÕES PARCIAIS

Este capítulo apresentou uma coleta de dados do tempo de inicialização e sincronização das arquiteturas IT-SPIRE e SPIRE, as quais foram implantadas na rede da Universidade do Estado de Santa Catarina. Além disso, foi realizada uma análise comparativa dos dados obtidos para ambas as arquiteturas.

A partir dos resultados apresentados conclui-se principalmente que:

- Tornar o Servidor SPIRE tolerante a intrusões adiciona uma média de 420,5 ms ao tempo de inicialização do SPIRE, mantendo o tempo de sincronização aproximadamente comparável.
- Os resultados obtidos para o IT-SPIRE apresentam variabilidade considerável. Parte dessa variabilidade pode ser herdada do SPIRE e de variações de desempenho do BFT-SMaRt.

5 CONCLUSÃO

O gerenciamento de identidades de software em ambientes dinâmicos e heterogêneos é um componente importante para a implementação de arquiteturas de confiança zero. O arcabouço SPIFFE e sua implementação de referência SPIRE oferecem uma solução de gerenciamento de identidades de software baseada na atestação de componentes, o que evita a necessidade de compartilhamento de segredos. A arquitetura do SPIRE possui alguns componentes críticos, que são considerados confiáveis; o comprometimento da integridade ou disponibilidade desses componentes pode afetar a segurança de uma aplicação que usa o SPIRE.

Partindo de uma análise dos componentes críticos do SPIRE e de como eles impactam a segurança de aplicações, este trabalho propõe o IT-SPIRE, um Servidor SPIRE tolerante a intrusões que tem por objetivo melhorar a resiliência da arquitetura SPIRE. A arquitetura do IT-SPIRE consiste em um conjunto de n Servidores que usam Replicação Máquina de Estados (RME) tolerante a falhas bizantinas: para tolerar f Servidores comprometidos, são necessárias $n \geq 3f + 1$ réplicas. O IT-SPIRE implementa RME por meio da biblioteca BFT-SMaRt (BESSANI; SOUSA; ALCHIERI, 2014). Com o intuito de minimizar as modificações no código do SPIRE, foram introduzidos dois *proxies* responsáveis por intermediar a comunicação entre Agentes e Servidores. Isso permite que o funcionamento interno do Agente, o seu papel na arquitetura, a interação entre as Cargas de Trabalho e Agente, e as identidades fornecidas ao Agente e às Cargas de Trabalho permaneçam inalterados, sendo necessárias apenas algumas alterações no funcionamento do Servidor SPIRE.

Para demonstrar a viabilidade da proposta, a arquitetura IT-SPIRE foi implementada na forma de protótipo. Foi também realizada uma avaliação experimental visando obter uma estimativa inicial do custo da proposta do ponto de vista do desempenho. Em relação à arquitetura SPIRE, o protótipo do IT-SPIRE apresentou um tempo médio de inicialização 420,5 ms maior, com tempos de sincronização aproximadamente comparáveis. Em ambos os casos, o tempo adicional obtido é condizente com a segurança extra fornecida pelo IT-SPIRE.

Como trabalhos futuros, vislumbram-se duas linhas de atuação. A primeira, de natureza mais prática, é evoluir a implementação do IT-SPIRE, indo de um protótipo em direção a uma versão apta a ser usada em ambientes de produção. A segunda é evoluir a arquitetura, projetando um mecanismo de recuperação para réplicas comprometidas e estendendo o IT-SPIRE para as arquiteturas de implantação Nested SPIRE e Federated SPIRE.

REFERÊNCIAS

AVIŽIENIS, Algirdas et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, Jan.–Mar. 2014. Citado na página 18.

BABAY, Amy et al. Deploying intrusion-tolerant SCADA for the power grid. In: **IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2019. p. 328–335. Citado na página 23.

BARRETO, Luciano; FRAGA, Joni; SIQUEIRA, Frank. An intrusion tolerant identity provider with user attributes confidentiality. **Journal of Information Security and Applications**, Elsevier, v. 63, p. 103045, 2021. Citado 3 vezes nas páginas 23, 33 e 35.

BARRETO, Luciano et al. An intrusion tolerant identity management infrastructure for cloud computing services. In: **IEEE 20th International Conference on Web Services (ICWS)**. [S.l.: s.n.], 2013. p. 155–162. Citado 3 vezes nas páginas 23, 33 e 35.

BAUM, Carsten et al. PESTO: Proactively secure distributed single sign-on, or how to trust a hacked server. In: **IEEE European Symposium on Security and Privacy (EuroS&P)**. Genoa, Italy: [s.n.], 2020. p. 587–606. Citado na página 23.

BESSANI, Alysson et al. DepSky: Dependable and secure storage in a cloud-of-clouds. **ACM Transactions on Storage**, v. 9, n. 4, p. 1–33, nov. 2013. Citado na página 23.

BESSANI, Alysson; SOUSA, João; ALCHIERI, Eduardo. State machine replication for the masses with BFT-SMaRt. In: **IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. [S.l.: s.n.], 2014. Citado 3 vezes nas páginas 44, 48 e 54.

BESSANI, Alysson Neves; ALCHIERI, Eduardo. A guided tour on the theory and practice of state machine replication. In: **Tutorial at the 32nd Brazilian symposium on computer networks and distributed systems**. [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 23 e 24.

BRANDÃO, Luís A. N.; MOUHA, Nicky; VASSILEV, Apostol. **Threshold Schemes for Cryptographic Primitives**. Gaithersburg, MD, 2019. Disponível em: <https://doi.org/10.6028/NIST.IR.8214>. Citado na página 22.

BRAUN, Johannes et al. CA trust management for the Web PKI. **Journal of Computer Security**, IOS Press, v. 22, n. 6, p. 913–959, 2014. Citado na página 30.

BUFFERS, Protocol. **Overview**. 2024. Protocol Buffers Documentation. Disponível em: <https://protobuf.dev/overview/>. Citado na página 27.

COCHAK, Henrique et al. Enhancing SPIFFE/SPIRE environment with a nested security token model. In: INSTICC. **Proceedings of the 14th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER**. [S.l.]: SciTePress, 2024. p. 184–191. ISBN 978-989-758-701-6. Citado 2 vezes nas páginas 32 e 33.

CORREIA, Miguel P. Serviços distribuídos tolerantes a intrusões: Resultados recentes e problemas abertos. In: **V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg) – Livro Texto dos Minicursos**. [S.l.]: SBC, 2005. p. 113–162. Citado 4 vezes nas páginas 18, 19, 21 e 22.

COSTA, Caio Yuri da Silva; ALCHIERI, Eduardo Adilio Pelinson. Diversity on state machine replication. In: IEEE. **2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)**. [S.l.], 2018. p. 429–436. Citado 3 vezes nas páginas 44, 45 e 53.

COSTA, Pedro A. R. S. et al. Medusa: An efficient cloud fault-tolerant MapReduce. In: **IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)**. Cartagena, Colombia: [s.n.], 2016. p. 443–452. Citado na página 23.

CRISTIAN, Flaviu. Understanding fault-tolerant distributed systems. **Communications of the ACM**, v. 34, n. 2, p. 56–78, fev. 1991. Citado na página 19.

DÉFAGO, Xavier; SCHIPER, André; URBÁN, Péter. Total order broadcast and multicast algorithms: Taxonomy and survey. **ACM Computing Surveys**, v. 36, n. 4, p. 372–421, dez. 2004. Citado na página 24.

DESWARTE, Yves; POWELL, David. Internet security: An intrusion-tolerance approach. **Proceedings of the IEEE**, v. 94, n. 2, p. 432–441, fev. 2006. Citado 2 vezes nas páginas 15 e 18.

FALCÃO, Eduardo et al. Supporting confidential workloads in SPIRE. In: **IEEE International Conference on Cloud Computing Technology and Science (CloudCom)**. [S.l.: s.n.], 2022. p. 186–193. Citado 3 vezes nas páginas 25, 32 e 33.

FELDMAN, Daniel et al. **Solving the Bottom Turtle – a SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity**. first. Nova Zelândia: Sprint Lab, 2020. ISBN 978–0–578–77737–5. Citado 8 vezes nas páginas 14, 24, 25, 27, 28, 30, 32 e 36.

FERRETTI, Luca et al. Survivable zero trust for cloud computing environments. **Computers & Security**, Elsevier, v. 110, p. 102419, 2021. Citado 2 vezes nas páginas 34 e 35.

FLORA, José. Improving the security of microservice systems by detecting and tolerating intrusions. In: **IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)**. Coimbra, Portugal: [s.n.], 2020. p. 131–134. Citado na página 23.

GEMMELL, Peter S. An introduction to threshold cryptography. **Cryptobytes—The Technical Newsletter of RSA Laboratories**, v. 2, n. 3, p. 7–12, Winter 1997. Disponível em: <https://networkdls.com/Articles/crypto2n3.pdf>. Citado na página 22.

GITHUB. **spire-api-sdk**. 2024. GitHub. Disponível em: <https://github.com/biareichert/spire-api-sdk>. Citado na página 45.

GRPC. **Why gRPC?** 2024. Disponível em: <https://grpc.io/>. Citado na página 27.

HADZILACOS, Vassos; TOUEG, Sam. **A Modular Approach to Fault-Tolerant Broadcasts and Related Problems**. 1994. Technical report TR 94-1425, Department of Computer Science, Cornell University. Citado 2 vezes nas páginas 23 e 24.

IBM. **Remote Authentication Dial In User Service overview**. 2023. IBM Documentation. Disponível em: <https://www.ibm.com/docs/en/i/7.3?topic=authentication-remote-dial-in-user-service-overview>. Citado na página 33.

JENSEN, Nikolas. Análise de segurança do Secure Production Identity Framework for Everyone (SPIFFE). **Trabalho de Conclusão do Curso (Graduação) - Universidade do Estado de Santa Catarina, Curso de Ciência da Computação,**, Joinville, SC, 2022. Disponível em: http://sistemabu.udesc.br/pergamumweb/vinculos/000098/00009853.pdf. Citado 2 vezes nas páginas 32 e 33.

JESSUP, Andrew et al. DVID: Adding delegated authentication to SPIFFE trusted domains. In: SPRINGER. **International Conference on Advanced Information Networking and Applications**. [S.l.], 2024. p. 289–300. Citado 2 vezes nas páginas 32 e 33.

JONES, Michael B.; BRADLEY, John; SAKIMURA, Nat. **JSON Web Token (JWT)**. [S.l.]: RFC Editor, 2015. RFC 7519. (Request for Comments, 7519). Disponível em: <https://rfc-editor.org/rfc/rfc7519.txt>. Citado na página 36.

KOSCHEL, Arne et al. A look at service meshes. In: **International Conference on Information, Intelligence, Systems & Applications (IISA)**. [S.l.: s.n.], 2021. p. 1–8. Citado na página 31.

KREUTZ, Diego et al. Towards secure and dependable authentication and authorization infrastructures. In: **IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)**. [S.l.: s.n.], 2014. p. 43–52. Citado 3 vezes nas páginas 23, 33 e 35.

KREUTZ, Diego et al. A cyber-resilient architecture for critical security services. **Journal of Network and Computer Applications**, v. 63, p. 173–189, 2016. ISSN 1084-8045. Citado 2 vezes nas páginas 34 e 35.

KSHEMKALYANI, Ajay D; SINGHAL, Mukesh. **Distributed Computing: Principles, Algorithms, and Systems**. [S.l.]: Cambridge University Press, 2011. Citado na página 19.

LAMPORT, Leslie; SHOSTAK, Robert; PEASE, Marshall. The Byzantine generals problem. **ACM Transactions on Programming Languages and Systems**, v. 4, n. 3, p. 382–401, jul. 1982. Citado na página 19.

MADAN, Bharat B. et al. Intrusion tolerant multi-cloud storage. In: **IEEE International Conference on Smart Cloud (SmartCloud)**. New York, NY: [s.n.], 2016. p. 262–268. Citado na página 23.

MAGNANINI, Federico; FERRETTI, Luca; COLAJANNI, Michele. Flexible and survivable single sign-on. In: **13th International Symposium on Cyberspace Safety and Security (CSS)**. [S.l.: s.n.], 2021. p. 182–197. Citado 3 vezes nas páginas 23, 34 e 35.

MICROSOFT. **Identity and access management (IAM) fundamental concepts**. 2024. Disponível em: <https://learn.microsoft.com/en-us/entra/fundamentals/identity-fundamental-concepts>. Citado na página 14.

OBELHEIRO, Rafael R.; BESSANI, Alysson Neves; LUNG, Lau Cheuk. Analisando a viabilidade da implementação prática de sistemas tolerantes a intrusões. In: **V Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)**. Florianópolis, SC: [s.n.], 2005. p. 99–112. Citado na página 22.

OBENSHAIN, Daniel et al. Practical intrusion-tolerant networks. In: **IEEE International Conference on Distributed Computing Systems (ICDCS)**. Nara, Japan: [s.n.], 2016. p. 45–56. Citado na página 23.

OPENID. **OpenID Foundation**. 2023. OpenID. Disponível em: <https://openid.net/>. Citado na página 33.

PONTES, Davi et al. Attesting AMD SEV-SNP virtual machines with SPIRE. In: **Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing**. [S.l.: s.n.], 2023. p. 1–10. Citado 2 vezes nas páginas 32 e 33.

POWELL, David. Failure mode assumptions and assumption coverage. In: RANDELL, Brian et al. (Ed.). **Predictably Dependable Computing Systems**. [S.l.]: Springer Berlin Heidelberg, 1995. p. 123–140. Citado na página 20.

REICHERT, B. M.; OBELHEIRO, R. R. **An Integrity-Focused Threat Model for Software Development Pipelines**. 2022. Disponível em: <https://arxiv.org/abs/2211.06249>. Citado na página 14.

RIGNEY, Carl et al. **Remote Authentication Dial In User Service (RADIUS)**. [S.l.]: RFC Editor, 2000. RFC 2865. (Request for Comments, 2865). Disponível em: <https://rfc-editor.org/rfc/rfc2865.txt>. Citado na página 33.

ROSE, Scott et al. **Zero Trust Architecture**. [S.l.], 2020. Disponível em: <https://doi.org/10.6028/NIST.SP.800-207>. Citado na página 14.

SCHNEIDER, Fred B. Implementing fault-tolerant services using the state machine approach: a tutorial. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 22, n. 4, p. 299–319, dec 1990. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/98163.98167>. Citado na página 23.

SCHNEIDER, Fred B.; ZHOU, Lidong. Implementing trustworthy services using replicated state machines. **IEEE Security & Privacy**, v. 3, n. 5, p. 34–43, set. 2005. Citado 3 vezes nas páginas 16, 22 e 23.

SHAN, Guohou et al. Poligraph: Intrusion-tolerant and distributed fake news detection system. **IEEE Transactions on Information Forensics and Security**, v. 17, p. 28–41, 2021. Citado na página 23.

SHIREY, Robert W. **Internet Security Glossary, Version 2**. [S.l.]: RFC Editor, 2007. RFC 4949. (Request for Comments, 4949). Disponível em: <https://rfc-editor.org/rfc/rfc4949.txt>. Citado na página 21.

SIRIWARDENA, Prabath; DIAS, Wajjakkara Kankanamge Anthony Nuwan. **Microservices security in action**. [S.l.]: Manning, 2020. Citado na página 14.

SPIFFE. **Configuring SPIRE**. 2019. SPIFFE Documentation. Disponível em: <https://spiffe.io/docs/latest/deploying/configuring/>. Citado 2 vezes nas páginas 29 e 44.

SPIFFE. **SPIFFE Concepts**. 2019. SPIFFE Documentation. Disponível em: <https://spiffe.io/docs/latest/spiffe-about/spiffe-concepts/>. Citado na página 24.

SPIFFE. **SPIRE Concepts**. 2019. SPIFFE Documentation. Disponível em: <https://spiffe.io/docs/latest/spire-about/spire-concepts/>. Citado 4 vezes nas páginas 14, 26, 27 e 28.

SPIFFE. **Scaling SPIRE**. 2021. SPIFFE Documentation. Disponível em: <https://spiffe.io/docs/latest/planning/scaling_spire/>. Citado 3 vezes nas páginas 29, 30 e 31.

SPIFFE. **SPIFFE Federation**. 2022. GitHub. Disponível em: <https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE_Federation.md>. Citado na página 30.

SPIFFE. **The SPIFFE Identity and Verifiable Identity Document**. 2022. GitHub. Disponível em: <https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE-ID.md>. Citado na página 25.

SPIRE. **v0.12.0**. 2020. GitHub. Disponível em: <https://github.com/spiffe/spire/releases/tag/v0.12.0>. Citado na página 26.

SQLITE. **What Is SQLite?** 2024. Disponível em: <https://www.sqlite.org/>. Citado na página 38.

STEEN, Maarten van; TANENBAUM, Andrew S. **Distributed Systems, 4th Ed**. [S.l.]: <https://distributed-systems.net>, 2023. Citado na página 19.

TRESTIOREANU, Lucian et al. SPON: Enabling resilient inter-ledgers payments with an intrusion-tolerant overlay. In: **IEEE Conference on Communications and Network Security (CNS)**. Tempe, AZ: [s.n.], 2021. p. 92–100. Citado na página 23.

VERÍSSIMO, Paulo E.; NEVES, Nuno F.; CORREIA, Miguel P. Intrusion-tolerant architectures: Concepts and design. In: LEMOS, Rogério de; GACEK, Cristina; ROMANOVSKY, Alexander (Ed.). **Architecting Dependable Systems**. [S.l.]: Springer Berlin Heidelberg, 2003. p. 3–36. Citado 4 vezes nas páginas 15, 20, 21 e 37.

VOELP, Marcus; VERÍSSIMO, Paulo E. Intrusion-tolerant autonomous driving. In: **IEEE International Symposium on Real-Time Distributed Computing (ISORC)**. Singapore: [s.n.], 2018. p. 130–133. Citado na página 23.

WANG, Yawen et al. Protecting scientific workflows in clouds with an intrusion tolerant system. **IET Information Security**, v. 14, n. 2, p. 157–165, 2020. Citado na página 23.