PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

JULIANO SARTORI LANGARO

AUTENTICAÇÃO NÃO-INTERATIVA COM GESTÃO DE SEGREDOS EM ARQUITETURA DE IAC COM ZERO TRUST

CURITIBA
2025

JULIANO SARTORI LANGARO

AUTENTICAÇÃO NÃO-INTERATIVA COM GESTÃO DE SEGREDOS EM ARQUITETURA DE IAC COM ZERO TRUST

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Altair Olivo Santin
Coorientador: Prof. Dr. Eduardo Kugler Viegas

CURITIBA

2025

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Gisele Alves – CRB 9/1578

Langaro, Juliano Sartori
L269a Autenticação não-interativa com gestão de segredos em arquitetura de IaC com
2025 Zero Trust / Juliano Sartori Langaro ; orientador : Altair Olivo Santin ; coorientador : Eduardo Kugler Viegas. – 2025.
138 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2025
Bibliografia: f. 128-138

1. Infrastructure as code. 2. Computadores – Medidas de segurança.
I. Santin, Altair Olivo. II. Viegas, Eduardo Kugler. III. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. IV. Título.

CDD. 20. ed. – 004

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

Curitiba, 25 de março de 2025.

51-2025

DECLARAÇÃO

Declaro para os devidos fins, que **JULIANO SARTORI LANGARO** defendeu a dissertação de Mestrado intitulada "**Autenticação não-interativa com Gestão de Segredos em Arquitetura de IaC com Zero Trust"**, na área de concentração Ciência da Computação no dia 20 de fevereiro de 2025, no qual foi aprovado.

Declaro ainda, que foram feitas todas as alterações solicitadas pela Banca Examinadora, cumprindo todas as normas de formatação definidas pelo Programa.

Por ser verdade, firmo a presente declaração.

gov.br
Documento assinado digitalmente
EMERSON CABRERA PARAISO
Data: 25/04/2025 09:51:36-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Dr. Emerson Cabrera Paraiso
Coordenador do Programa de Pós-Graduação em Informática

AGRADECIMENTOS

Agradeço à Pontifícia Universidade Católica do Paraná (PUCPR) pela estrutura e ambiente de aprendizado proporcionados durante o meu mestrado.

Ao Programa de Pós-Graduação em Informática (PPGIa), pela concessão do auxílio da comissão de bolsas. Este projeto foi parcialmente financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), por meio das bolsas nº 304990/2021-3 e 407879/2023-4.

Ao Prof. Dr. Altair Olivo Santin, pela confiança depositada, pelas valiosas oportunidades durante o mestrado, pelos ensinamentos imprescindíveis e pela paciência ao longo dos desafios enfrentados.

Ao Prof. Dr. Eduardo Kugler Viegas, pela parceria e orientações em cada etapa deste processo.

Aos professores do PPGIa, por compartilharem seu conhecimento e por contribuírem para minha formação acadêmica.

Aos colegas de laboratório e amigos da PUCPR – Adilson, Eloísa, Franciscon, Bruno, Miguel, Jonathan e Pedro – pela parceria, apoio e troca de experiências.

Aos amigos Fellipe Veiga e Juarez de Oliveira, pela grande parceria nos momentos decisivos da minha pesquisa, principalmente em compartilhar conhecimentos técnicos; sem vocês, este caminho teria sido infinitamente mais desafiador. Muito obrigado!

Ao coordenador da PUCPR, Diogo Xavier Saes, pela constante parceria e incentivo durante toda a jornada.

Um agradecimento especial ao meu grande amigo Carlos Henrique Sopchaki, pelo apoio inestimável e pelas contribuições ao longo do caminho.

Aos amigos que sempre me encorajaram, em especial João Alberto Wyzykowski e Claudio Renato Fogazzi, pela amizade e motivação constantes.

À minha família, pelo amor, paciência e apoio incondicional. Em especial, ao meu filho Davi, minha mãe Lígia e minha companheira Jayana, cuja presença e incentivo foram determinantes para alcançar esta conquista.

Por fim, a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho, deixo a minha mais profunda gratidão.

“Vivendo, se aprende;
mas o que se aprende, mais,
é só a fazer outras maiores perguntas”.

João Guimarães Rosa
Grande Sertão: Veredas

RESUMO

Em modelos tradicionais, códigos autônomos como *scripts* e componentes conectados, geralmente armazenam segredos (credenciais) estaticamente no código. Esse método expõe os segredos a riscos, pois um acesso não autorizado ao código pode comprometer as credenciais. Apesar de a criptografia ser uma alternativa, surgem desafios com relação ao armazenamento das chaves (segredos) em arquivos locais de forma segura em processos não-interativos, onde não há um *hardware* dedicado a este fim. Para enfrentar esse desafio, propomos uma arquitetura *Zero Trust* baseada em infraestrutura como código (*IaC)*, que aplica restrições de acesso aos códigos e credenciais armazenados ou em execução. Nossa proposta utiliza técnicas de ofuscação, complementadas por camadas de segurança como ZTNA, que incorpora práticas de microssegmentação e autenticação de dispositivos e usuários. Esse modelo exige que um atacante viole múltiplas camadas de proteção, como o ZTNA, algoritmos de ofuscação e códigos rodando em memória, para ter chance de êxito. A eficácia da abordagem foi validada por meio do modelo do adversário e testes de invasão (*pentest*), mostrando que a arquitetura proposta oferece resiliência a ataques.

Palavras-chave: *Zero Trust;* infraestrutura como código; gestão de segredos; autenticação não-interativa.

ABSTRACT

In traditional models, autonomous code such as scripts and connected components usually store secrets (credentials) statically in the code. This method exposes the secrets to risk, as unauthorized access to the code can compromise the credentials. Although cryptography is an alternative, challenges arise with regard to storing keys (secrets) securely in local files in non-interactive processes, where there is no dedicated hardware for this purpose. To address this challenge, we propose a Zero Trust architecture based on infrastructure as code (IaC), which applies access restrictions to stored or running code and credentials. Our proposal uses obfuscation techniques, complemented by security layers such as ZTNA, which incorporates micro-segmentation and device and user authentication practices. This model requires an attacker to breach multiple layers of protection, such as ZTNA, obfuscation algorithms and code running in memory, in order to have a chance of success. The effectiveness of the approach was validated using the adversary model and pentests, showing that the proposed architecture offers resilience to attacks

Keywords: Zero Trust; infrastructure as code; secrets management; non-interactive authentication.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AES	Advanced Encryption Standard
API	Application Programming Interface
ARM	Azure Resource Manager
CWE	Common Weakness Enumerations
CIS	Center for Internet Security
CLI	Command-Line Interface
CPU	Central Unit Processor
CSRF	Cross Site Request Forgery
CVE	Common Vulnerabilities and Exposures
DEVOPS	Development and Operations
DOS	Denail of Service
GCP	Google Cloud Plataform
GLPI	Gestionnaire Libre de Parc Informatique
HCL	Hashicorp Configuration Language
HTTP	Hypertext Transfer Protocol
HTTS	Hypertext Transfer Protocol Secure
HOTP	Time-based One-Time Password
HMAC	Hash-based Message Authentication Code
HVAC	Hashicorp Vault API client
IAM	Identity and Access Management
IaC	Infrastructure as Code
ICMP	Internet Control Message Protocol
IdP	Identity Provider
IDS	Intrusion Detection System
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code
MFA	Multiple Factors-Authentication
NMAP	Network Mapper
NIST	National Institute of Standards and Technology
OIDC	OpenID Connect
OTP	One Time Password
OWASP	Open Worldwide Application Security Project
PAM	Privileged Acess Management
PyOTP	Python One-Time Password Library
RFC	Request for Comments
SAAS	Software as a Service
SAML	Security Assertion Markup Language
SASE	Secure Access Service Edge
SCIM	System for Cross-Domain Identity Management
SDN	Software Defined Network
SGX	Intel Software Guard Extension
SIEM	Security Information and Event Management

SMS	Short Message Service
SP	Service Provider
SSH	Secure Shell
TEE	Trusted Execution Environment
TCP	Transmission Control Protocol
TI	Information Technology
TOTP	HMAC-based One-Time Password
UDP	User Datagram Protocol
UI	User Interface
VPC	Virtual Private Cloud
VPN	Virtual Private Network
XSS	Cross-site scripting
YAML	Yet Another Markup Language
ZT	Zero Trust
ZTNA	Zero Trust Network Access

SUMÁRIO

Capítulo 1

Introdução

A infraestrutura de tecnologia da informação, composta por *hardware* e *software*, tornou-se mais complexa e distribuída com a popularização da computação em nuvem e ambientes híbridos, introduzindo novos desafios de segurança, como o armazenamento seguro de credenciais em processos de autenticação não-interativa.

Esse processo pode ser automatizado por meio da abordagem da infraestrutura como código, conhecida como *IaC* (*Infrastructure as Code*), que permite a criação, manutenção e atualização de ambientes de forma orquestrada, assegurando consistência, agilidade e controle sobre mudanças e atualizações no ambiente de TI.

A adoção da *IaC*, baseada nos mesmos princípios do ciclo de desenvolvimento de *software*, pode oferecer às equipes de TI a possibilidade de automatizar e implementar ambientes computacionais que se mostrem eficientes e dentro de validações de conformidade. Embora amplamente utilizada em nuvens públicas e ambientes híbridos, a abordagem pode ser aplicada em ambientes *on-premise*, garantindo padronização e automação da infraestrutura em qualquer cenário, seja local ou na nuvem. Essa flexibilidade promove maior consistência na gestão dos recursos.

Segundo Opdebeeck, Zerouali e Roover (2023), a *IaC* consiste na prática de desenvolver e gerenciar infraestruturas de computação utilizando código-fonte executável. Contudo, essa abordagem introduziu novos vetores maliciosos, ampliando os riscos associados à segurança.

Pahl *et al*. (2017) destacam que os *scripts IaC* auxiliam no provisionamento e gerenciamento de uma infraestrutura baseada em nuvem. Contudo, ao desenvolver esse código de configuração, profissionais de TI podem introduzir *Security Smells*, padrões de codificação que indicam potenciais falhas de segurança (Rahman *et al*. 2019).

No contexto da segurança, esse risco é potencializado pelo fato de que os *scripts IaC* frequentemente são criados ou modificados por desenvolvedores de *software,* e não por especialistas em segurança, resultando em um maior risco de configurações inadequadas (Ahmed, 2019).

No contexto da segurança de TI, ganha destaque o modelo *Zero Trust*, que busca aperfeiçoar a proteção de dados sensíveis em contraposição à segurança tradicional baseada em perímetro. Essa abordagem considera todo o tráfego como não confiável por padrão, independentemente de sua origem. Controles de acesso são aplicados com base na identidade do usuário ou dispositivo, no contexto da solicitação e na postura de segurança do dispositivo (Adaptado de Cao *et al.,* 2024).

Este modelo oferece uma abordagem aprimorada em comparação aos métodos tradicionais, os quais, mesmo utilizando *firewalls*, VPNs e sistemas de detecção/prevenção de intrusões (IDS/IPS), podem enfrentar desafios ao lidar com agentes maliciosos cada vez mais sofisticados.

Adaptado de Syed *et al*. (2022), o princípio básico do *Zero Trust* é reforçar a autenticação e o controle de acesso, aplicando várias camadas de proteção tanto para dispositivos quanto para servidores. A adoção desse modelo implica uma mudança de paradigma, onde nenhum componente, seja interno ou externo à rede, é confiável por padrão. Essa abordagem visa mitigar as ameaças advindas de práticas convencionais, garantindo maior proteção contra ataques sofisticados e violações de dados.

Paralelamente, o uso de *IaC* para automação de infraestrutura oferece consistência e agilidade, mas também introduz desafios relacionados à segurança de segredos e credenciais sensíveis, presentes nos *scripts*. Esses segredos, como senhas e chaves, exigem práticas robustas de gerenciamento para evitar exposições indevidas (Rahman *et al*., 2021a).

Além disso, práticas de DevOps também demandam atenção, pois incluem ferramentas e métodos para gerenciar credenciais sensíveis, como senhas, APIs e *tokens*, em todo o ecossistema de TI (Adaptado de Alonso; Piliszek; Cankar, 2023).

De acordo com um relatório da IBM, o custo médio global de uma violação de dados em 2023 foi de USD 4,45 milhões, um aumento de 15% em três anos. Esses dados

reforçam a importância de incorporar práticas de segurança robustas, como *Secure by Design* e *Zero Trust Security*, para lidar com os desafios contemporâneos da cibersegurança (IBM, 2023a).

Esta pesquisa propõe uma arquitetura de segurança baseada nos princípios do modelo *Zero Trust*, integrando automação de infraestrutura por *IaC* e técnicas avançadas para proteção de credenciais em processos não interativos. A abordagem utiliza um mecanismo de autenticação que combina *One-Time Passwords* (OTPs) e *Hash-based Message Authentication Codes* (HMACs) para garantir a integridade e autenticidade das comunicações, ao mesmo tempo que protege segredos por meio de ofuscação.

Por meio dessa integração, espera-se demonstrar que é possível aprimorar a segurança de sistemas computacionais. O objetivo é reduzir a superfície de ataque, dificultando a exploração por agentes maliciosos e promovendo um modelo de acesso alinhado às premissas do *Zero Trust*.

1.1 Problema e motivação

Nos últimos anos, a crescente dependência de sistemas computacionais em ambientes corporativos e governamentais destacou a necessidade de arquiteturas mais seguras e resilientes contra ciberataques. Com a popularização da computação em nuvem e a expansão de ambientes híbridos, a infraestrutura de tecnologia da informação tornou-se mais complexa e distribuída, e consequentemente, mais vulnerável.

A automação da infraestrutura de TI, viabilizada por provedores em nuvem e pela prática da *IaC,* introduziu novos desafios de segurança. Entre eles, destaca-se o armazenamento seguro de credenciais em processos de autenticação não-interativa, isto é, a validação de credencias entre sistemas sem a intervenção humana. O problema está em como garantir a proteção desses segredos, evitando que sejam armazenadas de forma inadequada, como em texto simples, APIs, variáveis de ambiente ou diretamente no código-fonte.

Conforme Rahman *et al.* (2022), segredos como identificadores de usuário, senhas, *tokens* de API e chaves de criptografia são elementos essenciais para autenticação entre artefatos de *software*. No entanto, a prática de inseri-los em texto claro, muitas vezes em sistemas de controle de versão, expõe o *software* a riscos de segurança. Credenciais armazenadas dessa forma tornam o *software* suscetível a violações, sendo este recurso amplamente reconhecido como uma fraqueza comum de segurança (CWE-798: Uso de credenciais embutidas – MITRE 2020).

Miller (2018) aponta que os principais riscos na gestão de segredos incluem a falta de visibilidade sobre credenciais privilegiadas, a utilização de segredos embutidos em código, a ausência de rotação de senhas e a dependência de processos manuais. Essas práticas aumentam significativamente o risco de comprometimento, em ambientes *on-premise* e na nuvem. Ainda que cofres de senhas ofereçam controle de acesso e rotatividade de credenciais, essas soluções podem apresentar lacunas de segurança, particularmente em sistemas baseados em perímetro.

Os modelos tradicionais de segurança baseados em perímetro são ineficazes contra movimentações laterais não autorizadas dentro da rede, permitindo que invasores explorem vulnerabilidades após um único ponto de comprometimento (Oliveira *et al*., 2022).

Nesse contexto, métodos de autenticação multifator (MFA), integração com APIs seguras como o WebAuthn, utilização de cofres de senhas e aplicação de práticas de ofuscação desempenham papéis primordiais na segurança. No entanto, esses métodos frequentemente requerem a interação humana durante o processo de autenticação, o que pode limitar a sua aplicação em cenários totalmente automatizados.

Para mitigar os riscos associados à autenticação não-interativa, é necessário adotar abordagens que eliminem dependências de segurança baseadas em perímetro. O modelo *Zero Trust* surge como uma alternativa promissora, pois considera todo o tráfego como não confiável por padrão. Essa abordagem aplica controles rigorosos com base na identidade, no contexto e na postura de segurança de usuários e dispositivos, reduzindo a exposição de credenciais e prevenindo acessos não autorizados.

A integração de princípios de *Zero Trust* com *IaC* pode fortalecer a segurança em ambientes computacionais, permitindo que a infraestrutura seja configurada de forma

consistente e segura. Embora ferramentas automatizadas, como Trivy e Checkov, desempenhem um papel importante na detecção de vulnerabilidades em *scripts IaC*, ainda há uma lacuna na implementação de controles *Zero Trust* combinados com técnicas.

Abordando o desafio do movimento lateral, os *firewalls* de perímetro oferecem *gateways* geralmente robustos, auxiliando na proteção contra invasores externos. No entanto, eles são menos eficazes na detecção e bloqueio de ataques originados dentro da própria rede e não conseguem proteger usuários fora do perímetro, como acessos remotos ou serviços baseados em nuvem (Oliveira *et al*., 2024).

A literatura carece de estudos que abordem detalhadamente a combinação de *IaC* com uma arquitetura *Zero Trust* integrada a mecanismos robustos de autenticação não-interativa, como o uso de HMAC e técnicas de ofuscação.

A segurança de credenciais em processos não-interativos exige a adoção de pressupostos fundamentais, como confidencialidade, integridade e autenticidade. Esses princípios são essenciais para proteger informações sensíveis contra acessos não autorizados e garantir a resiliência do sistema frente a ameaças modernas.

Ante esse cenário, esta pesquisa propõe uma abordagem que alia os princípios de *Zero Trust* à automação por *IaC*, incorporando um mecanismo de segurança que utiliza OTP (*One-Time Password*) e HMAC para autenticação em processos não-interativos. Essa abordagem pretende reforçar a proteção de credenciais sensíveis por meio de ofuscação, eliminando a necessidade de armazenamento em texto claro e reduzindo a superfície de ataque.

Além de abordar os desafios tradicionais da segurança em *IaC*, a pesquisa busca explorar como a aplicação integrada de *Zero Trust*, técnicas de ofuscação e autenticação adicional pode oferecer um modelo seguro e eficiente para ambientes computacionais. Essa proposta contribui diretamente para mitigar riscos associados à exposição de segredos, oferecendo um controle mais rigoroso sobre o acesso a informações sensíveis.

Em síntese, esta pesquisa visa implementar uma arquitetura de segurança baseada em *Zero Trust* utilizando *IaC*, alinhada a mecanismos de autenticação não-interativa. A integração de técnicas como ofuscação de credenciais e autenticação baseada em OTP e

HMAC pretende fortalecer a segurança contra ameaças e vulnerabilidades, estabelecendo um modelo eficiente e alinhado às demandas da cibersegurança moderna.

1.2 Objetivos

A pesquisa tem como objetivo propor um modelo de arquitetura *Zero Trust* utilizando *IaC*, voltado para autenticação não-interativa com ofuscação de segredos.

Para alcançar esse objetivo geral, foram definidos os seguintes objetivos específicos:

a) Implementar uma arquitetura *Zero Trust*, utilizando *IaC* para provisionar, configurar e manter o ambiente computacional, garantindo consistência e controle sobre os recursos;

b) Desenvolver um mecanismo de segurança com ofuscação de credenciais, para autenticação não-interativa baseada em OTP e HMAC, para proteger segredos e mitigar riscos de exposição;

c) Implementar, testar e avaliar o protótipo da proposta, validando sua eficácia na redução de vulnerabilidades e na proteção de sistemas computacionais contra ataques e acessos não autorizados.

1.3 Contribuições

Entre as principais contribuições desta dissertação para enfrentar os desafios relacionados ao gerenciamento seguro de credenciais, o método proposto aprimora os seguintes aspectos:

a) Autenticação segura em processos não-interativos, utilizando um mecanismo baseado em OTP e HMAC, para garantir a integridade e a autenticidade das

mensagens, mitigando riscos associados ao roubo de credenciais e acessos indevidos;

b) Armazenamento seguro de segredos por meio da técnica de ofuscação de credenciais, eliminando o risco de exposição em texto claro e mitigando vulnerabilidades associadas, garantindo que as credenciais não estejam disponíveis em sua forma original (texto em claro);
c) Modelo de arquitetura *Zero Trust* com *IaC*, que usa credenciais para prover o isolamento dos vários serviços e da infraestrutura *IaC* − flexível por ser programável, mas segura pelos vários níveis de controle *Zero Trust*, que evitam principalmente movimento lateral, roubo de credencias e escalação de privilégios.

Com base nas contribuições desta pesquisa, foi desenvolvido e submetido um artigo de conferência, que atualmente se encontra em processo de revisão para a 30th *IEEE Symposium on Computers and Communications* (ISCC), a ser realizada de 2 a 5 de julho de 2025, em Bolonha, Itália.

1.4 Estrutura do documento

O documento está organizado da seguinte forma, proporcionando uma visão clara e sequencial dos aspectos explorados na pesquisa:

- **Capítulo 1:** Introdução, que apresenta o contexto da pesquisa, a definição do problema, a motivação, os objetivos estabelecidos e as contribuições esperadas.
- **Capítulo 2:** Fundamentação teórica, abordando os principais conceitos, tecnologias e bases teóricas que sustentam o desenvolvimento da proposta.
- **Capítulo 3:** Trabalhos relacionados, destacando estudos anteriores relevantes e comparando-os com a abordagem proposta nesta pesquisa.
- **Capítulo 4:** Metodologia, explicando o método adotado, os passos seguidos e as ferramentas utilizadas para a realização da pesquisa.

- **Capítulo 5:** Implementação, detalhando a construção do protótipo e integrando as soluções propostas no ambiente controlado para validação.
- **Capítulo 6:** Avaliação e análise, apresentando os resultados obtidos e a interpretação detalhada do desempenho e da eficácia da proposta.
- **Capítulo 7:** Conclusão, com as considerações finais sobre a pesquisa, os aprendizados obtidos e as perspectivas para trabalhos futuros.

Capítulo 2

Fundamentação

Este capítulo aborda os conceitos fundamentais que sustentam esta pesquisa, incluindo Infraestrutura como Código (*IaC*), computação em nuvem e *containers*, DevOps (desenvolvimento e operações), mecanismos de segurança, MFA (múltiplos fatores de autenticação), ofuscação de código, criptografia, gestão de identidades, segredos e cofres de senhas, além dos princípios e abordagens do modelo *Zero Trust*. Também são explorados temas como ambientes seguros em *IaC*, segurança baseada em perímetro, movimento lateral, segmentação de rede, modelo de adversário e testes de invasão (*pentest*).

No contexto da pesquisa, a arquitetura proposta não só protege as aplicações e os sistemas envolvidos, mas também garante a segurança da própria infraestrutura automatizada, implementada por meio do *IaC*. Ao aderir aos princípios do modelo *Zero Trust*, o *IaC* não apenas configura e provisiona os recursos de forma consistente, mas também se torna parte integrante da estratégia de segurança, incorporando políticas rigorosas de controle e verificação contínua.

Além disso, será desenvolvido um mecanismo de segurança que combina ofuscação de credenciais e autenticação não-interativa, utilizando OTP e HMAC. Esse mecanismo visa reforçar a segurança das comunicações entre os sistemas e mitigar riscos associados ao comprometimento de segredos, alinhando-se às premissas do *Zero Trust*.

2.1 Infraestrutura como código

A computação em nuvem ampliou significativamente a quantidade de nós de infraestrutura necessários para suportar sistemas modernos (Nygard, 2007). Além disso, sistemas são lançados em produção com maior frequência, em muitos casos várias vezes ao dia, o que resulta em mudanças constantes na infraestrutura (Humble; Farley, 2010).

Nesse contexto, a *IaC* surge como uma abordagem essencial para gerenciar a complexidade, garantindo consistência e previsibilidade na configuração dos ambientes computacionais.

A *IaC* permite que ambientes sejam provisionados com especificações consistentes, pois as configurações são definidas em arquivos de código que podem ser versionados, revisados e reutilizados (Morris, 2016).

Segundo Wang (2022), a *IaC* substitui a configuração manual pela automação baseada em código, utilizando ferramentas e scripts especializados para provisionar e gerenciar recursos de TI. Essa abordagem não apenas simplifica operações, mas também fortalece a segurança, pois elimina a inconsistência e os erros comuns em processos manuais.

2.1.1 Classificações de *IaC*

Os princípios da *IaC* podem ser classificados em categorias que abordam diferentes aspectos da gestão de infraestrutura:

a) **Infraestrutura:** Refere-se à base tecnológica composta por *hardware* e *software*, como servidores, dispositivos de rede, *firewalls* e roteadores.
b) **Provisionamento de infraestrutura:** Automatiza a criação e o gerenciamento de recursos físicos e virtuais, como servidores e redes. Exemplos de ferramentas incluem Terraform, TOSCA e Cloudify.

c) **Gerenciamento de configuração:** Automatiza a instalação e atualização de *software* e serviços. Ferramentas como Ansible, Chef e Puppet são amplamente utilizadas.
d) ***Image building*:** Cria imagens de sistema operacional e aplicativos que podem ser replicadas para instanciar novos servidores. Exemplos incluem HashiCorp Packer e Docker.
e) **Orquestração:** Coordena múltiplas tarefas automatizadas, permitindo fluxos de trabalho complexos. A saída de uma tarefa pode ser usada como entrada para outra, como provisionar uma rede antes de configurar um servidor.

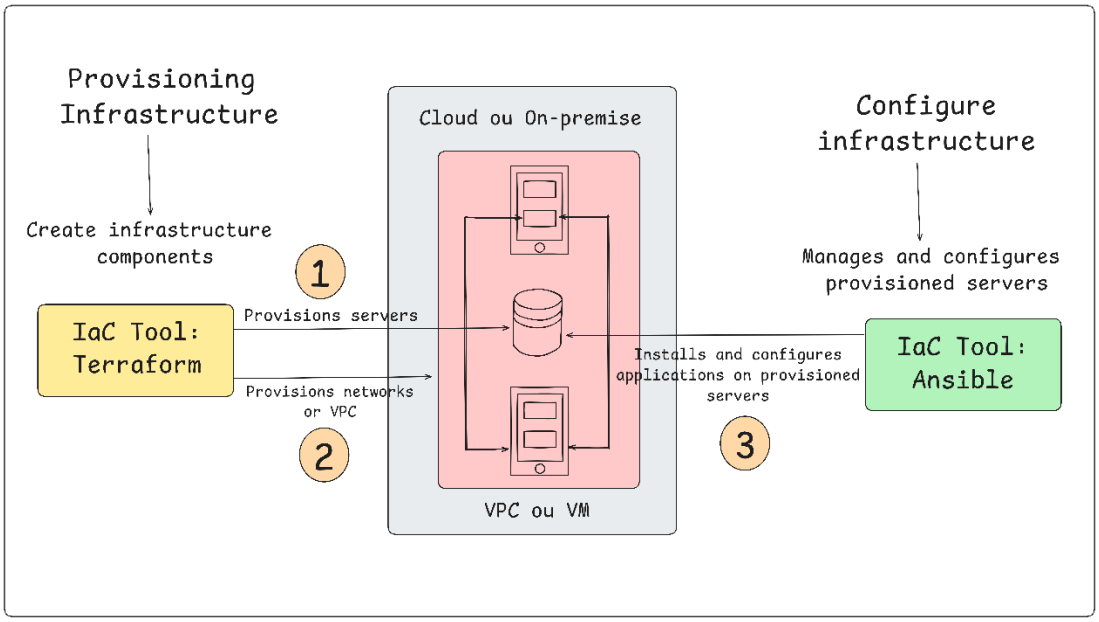
Com essas categorias, a *IaC* aplica os mesmos fluxos de trabalho do desenvolvimento de *software* à gestão de infraestrutura, incluindo versionamento, automação e testes. Essa abordagem garante consistência no provisionamento de ambientes, reduzindo erros humanos e acelerando processos (Adaptado de Wilson, 2023).

2.1.2 Benefícios e ferramentas de IaC

Um dos principais benefícios da *IaC* é a idempotência, que assegura que a execução repetida do código sempre resulte no mesmo estado da infraestrutura. Alterações manuais são automaticamente revertidas para o estado desejado, definido no código (Morris, 2016). Ferramentas como Terraform, Pulumi e Ansible são amplamente utilizadas para provisionamento e gerenciamento de configurações, garantindo independência de fornecedores e padronização.

A Figura 1 ilustra as diferenças entre ferramentas de provisionamento − como Terraform, que cria redes e servidores −, e ferramentas de gerenciamento de configuração, como Ansible, que configura aplicativos e serviços nos recursos provisionados.

Figura 1 – Diferença entre os processos de provisionar e configurar *IaC*



Fonte: Adaptado de Wilson (2023).

O fluxo de trabalho de *IaC* centra-se no desenvolvedor, com práticas de integração contínua e testes automatizados para validar a conformidade da infraestrutura com as especificações declaradas no código. Isso a torna uma base sólida para implementar os princípios do modelo *Zero Trust*, promovendo consistência e segurança em escala.

2.1.3 Computação em nuvem e *containers*

A computação em nuvem tornou-se indispensável para equipes de desenvolvimento e operações (DevOps), permitindo a redução de custos e o aumento da eficiência operacional (Ibrahim; Yousef; Medhat, 2022). Ao invés de adquirir e gerenciar servidores locais, as empresas podem utilizar serviços de nuvem para provisionar apenas os recursos necessários, reduzindo o número de profissionais necessários para a administração da infraestrutura.

Essa flexibilidade, aliada à automação proporcionada pela *IaC*, acelera o provisionamento de recursos como servidores, balanceadores de carga e bancos de dados, eliminando operações manuais repetitivas.

Ao contrário das infraestruturas locais, onde as equipes de TI possuem controle total sobre *hardware* e *software*, a nuvem transfere parte desse controle para os provedores de serviços. Isso cria um modelo compartilhado de responsabilidade pela segurança, onde os provedores cuidam da infraestrutura subjacente, enquanto as organizações são responsáveis pela configuração segura de seus ambientes (Red Hat, 2024).

A computação em nuvem proporciona um modelo de entrega escalável e sob demanda, permitindo um rápido crescimento de aplicações e infraestrutura. Porém, também introduz desafios relacionados à segurança, como visibilidade limitada e maior superfície de ataque.

Nesse contexto, os *containers* se apresentam como elementos de segurança e portabilidade e são amplamente utilizados em conjunto com a computação em nuvem devido à sua capacidade de empacotar e isolar aplicativos, juntamente com seus ambientes de execução (Garg; Garg, 2019). Isso facilita a movimentação de aplicações entre ambientes de desenvolvimento, teste e produção, garantindo consistência funcional. Além disso, *containers* permitem segmentação e isolamento, alinhando-se aos princípios do modelo *Zero Trust*.

De acordo com Jin (2021), a crescente necessidade de ciclos de desenvolvimento mais curtos, integração contínua e entrega contínua, assim como economia de custos nas infraestruturas de nuvem levaram ao surgimento de contêineres, que são mais confiáveis e eficientes do que a tecnologia de virtualização tradicional.

2.1.4 Ambientes de *IaC* seguros

Os profissionais utilizam scripts de infraestrutura como código (*IaC*) para provisionar servidores e configurar ambientes de desenvolvimento. No entanto, ao criar esses scripts, é possível que introduzam inadvertidamente "cheiros de segurança" — padrões recorrentes de codificação que indicam fragilidades de segurança e podem resultar em potenciais violações (Rahman *et al.,* 2019).

A empresa Gartner afirma que 99% das falhas de segurança da nuvem até 2025 será culpa dos clientes. Antes da *IaC* ser provisionada no ambiente de produção, as soluções de segurança *IaC* irão inspecioná-lo em busca de erros de configuração e problemas de segurança que possam deixá-lo vulnerável a ataques (Check Point, c1994-2025a).

No entanto, como o *IaC* tem o objetivo de descrever implantações físicas, ele também traz diversas preocupações relacionadas à segurança e outros possíveis problemas nos *scripts IaC*, que podem causar danos significativos, incluindo consequências graves em domínios e cenários de uso sensíveis. Por essa razão, garantir a qualidade dos *scripts IaC* e realizar inspeções de segurança são etapas fundamentais nos fluxos de trabalho que adotam os princípios do DevOps (Alonso; Piliszek; Cankar, 2023).

A segurança em práticas de *IaC* envolve diversos desafios e boas práticas que visam mitigar riscos no provisionamento automatizado de infraestrutura. Um dos principais aspectos da segurança em *IaC* é garantir que segredos, como senhas e chaves de API, não sejam *hardcoded* nos *scripts*, evitando a sua exposição acidental e reduzindo o risco de brechas. Ferramentas de detecção de segredos e varreduras contínuas de segurança ajudam a prevenir esse problema, além de impedir o uso de credenciais expostas (Adaptado de Queen, 2024).

De acordo com Queen (2024), os principais desafios incluem:

a) **Vulnerabilidades de imagem:** Modelos de imagens não confiáveis podem introduzir falhas em toda a infraestrutura.
b) **Desvios de configuração:** Alterações manuais ou não intencionais podem comprometer a segurança da infraestrutura.
c) **Gerenciamento de segredos:** O uso inadequado de *tokens*, chaves SSH e senhas aumenta o risco de exploração.

As ferramentas de varredura de vulnerabilidades são amplamente utilizadas para reforçar a segurança em códigos *IaC*. As principais ferramentas disponíveis estão apresentadas no Quadro 1, sendo que na pesquisa foram utilizados o Checkov e Trivy.

Quadro 1- Ferramentas de segurança para *IaC*

Ferramenta	Finalidade	Provedores Compatíveis	Descrição e Destaques
Checkov	Varredura de vulnerabilidades em *IaC*	AWS, Azure, GCP, Kubernetes	Detecta configurações inseguras em arquivos Terraform, CloudFormation, Kubernetes, entre outros. Suporta políticas customizadas de segurança. Ideal para integrações em *pipelines* CI/CD.
Tfsec	Varredura de vulnerabilidades em *IaC*	AWS, Azure, GCP	Especializado em varredura de arquivos Terraform, identifica falhas de segurança, configurações fracas e não conformidades com padrões como CIS e NIST. Pode ser integrado a *pipelines* de CI/CD.
Snyk *IaC*	Varredura de vulnerabilidades em *IaC*	AWS, Azure, GCP, Kubernetes	Detecta riscos em infraestrutura como código (Terraform, CloudFormation, Kubernetes). Oferece uma interface *web* e integração com ferramentas de DevOps. Focado em conformidade contínua e segurança automatizada.
Aqua Trivy	Varredura de vulnerabilidades em *containers* e *IaC*	AWS, Azure, GCP, Kubernetes	Além de analisar imagens de *containers*, realiza varredura em arquivos de configuração *IaC*, como Terraform e Kubernetes *Manifests*, detectando vulnerabilidades e configurações erradas.
Tenable.io	Varredura de vulnerabilidades em ativos e *IaC*	Multiplataforma	Plataforma abrangente de gestão de vulnerabilidades que inclui análise de arquivos *IaC*. Oferece suporte para detectar configurações inseguras e vulnerabilidades em ambientes de nuvem.

Fonte: Do autor.

Implementar ferramentas de varredura de vulnerabilidades, como Checkov e Trivy, pode ajudar a identificar falhas antes que a infraestrutura seja provisionada. Além disso, políticas de segurança devem ser integradas em todas as etapas do ciclo de vida de desenvolvimento, alinhando-se aos princípios do modelo *Zero Trust*.

2.2 Confiança zero (*Zero Trust*)

Em resposta às vulnerabilidades das soluções atuais de segurança de rede, o modelo de confiança zero segue a ideia de que nenhuma rede − seja interna ou externa − é confiável. O conceito de confiança zero está recebendo cada vez mais atenção tanto na pesquisa quanto na prática devido à sua promessa de cumprir novos requisitos complexos de segurança de rede. Apesar das vantagens da confiança zero sobre as soluções tradicionais, ela ainda não conseguiu substituir as abordagens existentes (Buck *et al*., 2021).

O NIST SP 800-207, *Zero Trust Architecture*, foi a primeira publicação a estabelecer as bases para a arquitetura *Zero Trust*. Ela apresenta o conceito de *Zero Trust* como um conjunto de princípios orientadores, em vez de tecnologias ou implementações específicas, além de incluir exemplos de arquiteturas baseadas nesse modelo.

No geral, a confiança zero não é uma tecnologia específica, mas uma abordagem holística que incorpora diferentes princípios básicos (Sultana *et al*., 2020). Assim, diferentes tecnologias (já existentes ou recentemente desenvolvidas) podem ser usadas para implementar a confiança zero. Os princípios básicos da confiança zero podem ser resumidos da seguinte forma (Rose *et al*., 2020):

- P1. Todas as fontes de dados e serviços computacionais são considerados recursos;
- P2. Toda comunicação é protegida independentemente de sua localização;
- P3. O acesso aos recursos individuais da organização é concedido para cada sessão;
- P4. O acesso aos recursos é determinado por uma política atualizada dinamicamente;
- P5. A organização monitora a integridade e segurança de todos os ativos;
- P6. Toda autenticação e autorização de recursos é dinâmica e estritamente aplicada antes que o acesso seja concedido;

- P7. A organização coleta o máximo possível de informações sobre o estado atual dos ativos, infraestrutura de rede e comunicações e as usa para melhorar a sua política de segurança.

Adaptado de Rose *et al*. (2020), de modo geral, o NIST SP 800-207 promove uma abordagem geral para a *Zero Trust* baseada nos princípios de privilégio mínimo, microssegmentação e monitoramento contínuo, incentivando as organizações a implementarem uma abordagem de segurança em camadas, que incorpore várias tecnologias e controles para se proteger contra ameaças.

Tradicionalmente, a segurança de redes seguia o modelo conhecido como "castelo e fosso", no qual a rede funcionava como um castelo protegido por um fosso. Apenas os usuários autorizados podiam atravessar o fosso e acessar os recursos internos da rede. Essa abordagem, conforme Fortinet (c2025), mostrou-se eficaz no combate a ameaças externas, mas falhou em lidar com ameaças internas já presentes na rede. Uma vez que um invasor atravessa esse perímetro inicial, ele tem acesso irrestrito aos recursos internos do sistema, o que pode resultar em sérias violações de segurança.

Conforme observado por Kindervag (2010), a abordagem tradicional de segurança requer um elemento de confiança implícita, o que pode ser explorado por atacantes para comprometer toda a infraestrutura de rede.

Conforme explicam Cao *et al*. (2024), o modelo de confiança zero também está intimamente relacionado à adoção de ferramentas de *IaC*, que permitem automatizar e reforçar as políticas de segurança em toda a infraestrutura de TI. A aplicação de políticas de segurança dinâmicas, conforme as boas práticas de *Zero Trust*, garante que apenas dispositivos e usuários autenticados possam acessar recursos sensíveis, minimizando significativamente os riscos de violações internas e externas.

Um equívoco comum entre as organizações é acreditar que a implementação do modelo de confiança zero é um objetivo simples e facilmente alcançável. Muitas vezes, presume-se que a aquisição de uma ferramenta ou solução específica seja suficiente para garantir a adoção completa do modelo. Contudo, essa abordagem não reflete a realidade, uma vez que o *Zero Trust* exige uma mudança estrutural e cultural que vai além da implementação de tecnologias isoladas.

Na presente pesquisa, a definição dos componentes mínimos necessários para caracterizar uma arquitetura *Zero Trust* será fundamentada na implementação de recursos por meio da infraestrutura como código (*IaC*). Essa abordagem permite validar configurações e assegurar que os padrões aplicados estejam aderentes aos princípios do *Zero Trust*, promovendo automação e conformidade de forma eficiente.

Por fim, é importante destacar que o modelo de confiança zero requer uma postura proativa e contínua de segurança, com verificações constantes de identidades, dispositivos e fluxos de acesso. Adaptado de Kindervag (2010), organizações que adotam esse modelo estão mais preparadas para enfrentar ameaças modernas e proteger seus dados em um ambiente cada vez mais conectado e vulnerável a ciberataques.

2.2.1 Arquitetura *Zero Trust* baseada em *IaC*

Em contrapartida aos modelos tradicionais de segurança baseados em perímetro, o *Zero Trust* adota uma abordagem centrada na verificação contínua de identidade e na análise da postura de segurança de usuários e dispositivos, independentemente de sua localização. Essa abordagem baseia-se na segurança orientada por contexto, onde o acesso é avaliado com base em fatores como identidade, localização, tipo de dispositivo, comportamento do usuário e risco associado a cada interação.

A arquitetura proposta aplica os princípios do *Zero Trust* para estabelecer uma estrutura de segurança robusta, tratando todas as interações, dentro ou fora da rede, como potencialmente inseguras até que possam ser devidamente validadas.

Para fortalecer a implementação da abordagem *Zero Trust*, a arquitetura é composta por diversas camadas de controle, incluindo políticas de acesso baseadas em identidade, autenticação multifator (MFA), uso de provedores de identidade (IdP) e análise de comportamento, que são práticas que aumentam a resiliência da arquitetura.

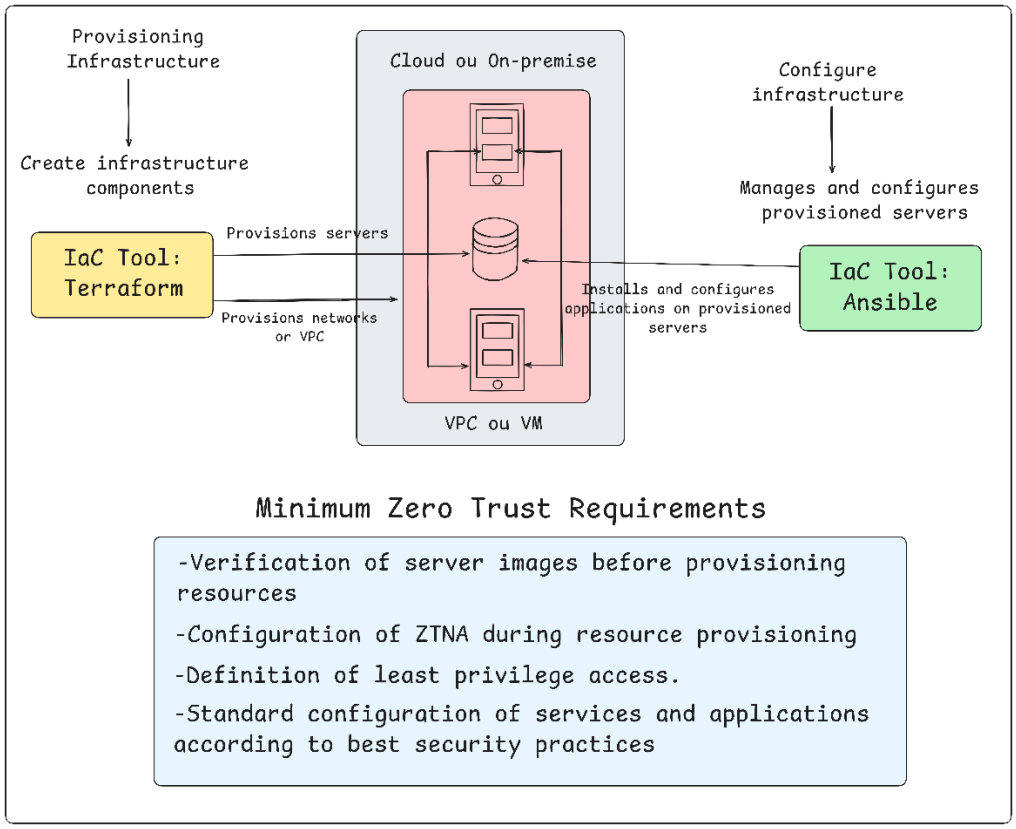
Cada uma dessas camadas fornece mecanismos adicionais de verificação, assegurando que as credenciais e os sistemas de autenticação sejam monitorados e validados de forma contínua, mantendo níveis de segurança e conformidade.

Uma arquitetura *Zero Trust* caracteriza-se por princípios básicos que devem estar presentes para atender aos requisitos mínimos desse modelo de segurança. O documento "*NIST SP 800-207, Zero Trust Architecture*" fornece uma definição de *Zero Trust* com um conjunto de princípios orientadores à adoção ao modelo (Adaptado de Rose *et al*., 2020).

Os princípios básicos incluem a verificação contínua, privilégio mínimo, monitoramento contínuo, acesso baseado em contexto e presumir a violação. Os princípios são parte integrante de uma arquitetura que pode ser caracterizada como *Zero Trust*, servindo como base para a sua implementação efetiva.

A Figura 2 apresenta os requisitos mínimos *Zero Trust* utilizados na arquitetura proposta, que facilita a visualização das camadas de segurança e/ou controles necessários para um modelo ser classificado como *Zero Trust*.

Figura 2 – Requisitos mínimos *Zero Trust* no modelo proposto



Fonte: Adaptado de Wilson (2023).

Os requisitos mínimos considerados nessa abordagem, conforme se apresenta na Figura 2, abrangem aspectos como controle de acesso baseado em identidade, monitoramento contínuo de dispositivos e aplicação de políticas dinâmicas de segurança.

A Figura 2 apresenta os requisitos mínimos alinhados com o modelo de segurança *Zero Trust*, descritos como:

1. **Verificação de imagens no servidor antes do provisionamento do recurso:**
   - Uso de ferramentas de validação estática de código (*IaC*) para garantir a segurança antes de disponibilizar o recurso.
2. **Configuração de ZTNA (*Zero Trust Network Access*) no provisionamento do recurso:**
   - Implementação de controles como autenticação MFA *(Multifactor Authentication*).
   - Microssegmentação por recurso.
   - Políticas de acesso granular baseadas em identidade.
   - Monitoramento contínuo e autenticação reforçada.
3. **Definição de privilégios mínimos:**
   - Garantia de que usuários e grupos possuem apenas os acessos necessários para suas funções, evitando permissões amplas ou padrão que possam comprometer a segurança.
4. **Configuração padrão de serviços e aplicativos conforme boas práticas de segurança:**
   - Uso obrigatório de HTTPS.
   - Alteração de senhas padrão.
   - Proibição do uso de portas inseguras, como a porta 80.
   - Utilização de cofres de senhas para gestão segura de credenciais.
   - Evitar a configuração padrão 0.0.0.0 e restringir o acesso a um endereço IP específico ou a uma faixa de IPs confiáveis.

Na seção 5.2 serão detalhados os componentes tecnológicos utilizados no modelo proposto, incluindo suas funções principais, integração com os princípios do modelo *Zero Trust* e o papel que desempenham na proteção do ambiente computacional.

A segunda abordagem que compõe a arquitetura é a utilização da *IaC,* que permite definir, configurar, provisionar e gerenciar a infraestrutura de forma automatizada em múltiplos provedores de nuvem, datacenters *on-premises,* entre outros serviços.

A *IaC* utiliza arquivos de configuração para descrever a infraestrutura desejada, garantindo consistência e reprodutibilidade na criação e gerenciamento de recursos, feito por *software*, dando suporte necessário para o *Zero Trust*.

A arquitetura assegura a proteção dos recursos provisionados pelo *IaC*, acessos e o próprio mecanismo de segurança para proteção de credenciais, viabilizando a implementação do modelo *Zero Trust*.

Essa abordagem adota uma postura de segurança detalhada, centrada na identidade e nas permissões de acesso. Ela vai além do uso de uma única tecnologia, priorizando a aplicação de controles rigorosos e personalizados para cada usuário, dispositivo ou interação. Assim, estabelece-se uma política de confiança mínima, garantindo que qualquer solicitação seja tratada como potencialmente insegura até ser devidamente validada.

É importante destacar que, apesar de desafios significativos, a adoção do modelo *Zero Trust* representa uma estratégia robusta para garantir a segurança em ambientes modernos. Embora a sua abordagem granular e orientada à identidade exija configurações detalhadas e uma gestão contínua que aumenta a carga operacional, seus benefícios superam as dificuldades.

A integração de tecnologias como MFA e IAM pode ser complexa, ainda mais em sistemas legados, e a necessária mudança cultural pode representar um obstáculo, assim como o custo inicial de implementação. No entanto, essas barreiras podem ser superadas com estratégias como automatização, adoção gradual e capacitação das equipes, garantindo uma segurança robusta e alinhada às exigências atuais de proteção contra ameaças.

2.3 Autenticação interativa e não-interativa

O objetivo da autenticação consiste em identificar as diversas entidades de um sistema computacional. Através da autenticação, o usuário interessado em acessar o sistema comprova que realmente é quem afirma ser (Maziero, 2019).

A autenticação é um dos principais mecanismos de segurança utilizados diariamente em sistemas computacionais para garantir que apenas usuários ou entidades

autorizadas possam acessar recursos protegidos (Stallings; Brown, 2015). Esse processo pode ser classificado em dois tipos principais: interativo e não-interativo.

A autenticação interativa exige a participação ativa de um ser humano no processo de validação. Nesse cenário, o usuário fornece informações, como nome de usuário e senha, utiliza dispositivos de autenticação multifatorial (MFA), ou se identifica por meio de métodos biométricos, como impressão digital, reconhecimento facial ou leitura de íris (NIST SP 800-63B, 2020).

Essa interação garante que o sistema possa confirmar que a pessoa que está tentando acessar é realmente quem afirma ser. Exemplos comuns de autenticação interativa incluem o *login* em sistemas operacionais, aplicações *web* e dispositivos móveis.

Por outro lado, a autenticação não-interativa ocorre sem a necessidade de uma intervenção direta de um usuário humano durante o processo de validação. Esse tipo de autenticação é amplamente utilizado em sistemas automatizados, onde aplicações, serviços e APIs se autenticam para estabelecer comunicações seguras. Ao invés de solicitar credenciais a um usuário, as informações de autenticação são previamente configuradas no sistema, como chaves de API, certificados digitais, *tokens* de acesso, ou *secrets* armazenados em cofres de senhas (RFC 6749).

É importante destacar que, em autenticações não-interativas, não há interação direta entre usuários e as entidades que realizam a autenticação. Ao invés disso, a comunicação ocorre diretamente entre os sistemas e serviços envolvidos. As credenciais são enviadas em trânsito ou acessadas em repouso, dependendo da configuração e da arquitetura adotada. Com isso, a proteção dessas credenciais se torna desafiadora, uma vez que qualquer comprometimento pode permitir que um atacante acesse recursos sensíveis, sem a necessidade de interação humana adicional.

Em sistemas que utilizam autenticação não-interativa, geralmente exigem o armazenamento persistente de credenciais. A autenticação não-interativa é fundamental para ambientes de computação em nuvem, microsserviços e sistemas de CI/CD (Integração contínua e Entrega contínua), onde diversas máquinas, *containers* e processos precisam estabelecer conexões seguras sem depender de um operador humano. Esse tipo

de autenticação é importante em cenários que envolvem a troca de informações sensíveis, garantindo a segurança e integridade das comunicações (Microsoft, 2023).

Um exemplo prático é o uso de uma chave de API (*API Key*) armazenada em um arquivo de configuração para permitir que um serviço *backend* se comunique com um serviço de terceiros, como um provedor de pagamento. Quando a aplicação faz uma requisição à API do provedor, a chave de API é enviada automaticamente no cabeçalho da requisição, autenticando a comunicação entre as duas partes. Nesse processo, não há qualquer interação humana para fornecer a chave de API durante cada requisição; ela já está configurada e é utilizada automaticamente pelo sistema.

Nesta pesquisa, o foco está na autenticação não-interativa, considerando sua relevância para soluções de segurança baseadas em arquiteturas *Zero Trust* e gestão de segredos. A proposta visa inserir mecanismos adicionais de validação, como o uso de OTPs baseados em HMAC, para reforçar a autenticação entre serviços, mitigando riscos de comprometimento de credenciais e aumentando a confiabilidade das comunicações entre componentes de sistemas distribuídos.

2.3.1 Pressupostos de segurança em autenticação não-interativa

Os pressupostos de segurança estabelecem as diretrizes fundamentais que orientam a arquitetura proposta, garantindo que as decisões de *design* estejam alinhadas às melhores práticas de segurança da informação. No contexto da autenticação não-interativa e gestão de segredos, consideram-se os seguintes princípios:

1. **Princípios fundamentais do modelo *Zero Trust***
    - Nenhuma entidade é confiável por padrão: Todo acesso deve ser verificado, independentemente da origem.
    - Privilégio mínimo: Os serviços devem operar com as permissões mínimas necessárias.
    - Autenticação e autorização contínuas: Cada solicitação de acesso deve ser validada dinamicamente.

- Presunção de violação: A arquitetura assume que sistemas podem ser comprometidos a qualquer momento, exigindo monitoramento contínuo.

2. **Segurança de credenciais e gestão de segredos**
   - Cofres de senhas: Credenciais não devem ser armazenadas em código-fonte ou arquivos de configuração.
   - Rotação periódica e expiração de segredos: Reduz o impacto de possíveis comprometimentos.
   - Autenticação baseada em OTPs e HMAC: Garante integridade e autenticação forte.

3. **Redução da superfície de ataque**
   - Ofuscação de credenciais: Protege segredos contra engenharia reversa.
   - Segmentação de rede: Impede movimentações laterais em caso de comprometimento de um serviço.
   - Proteção contra ataques de repetição: Uso de OTPs dinâmicos para evitar reutilização de *tokens* interceptados.
   - Esses pressupostos estabelecem a base para a arquitetura proposta e serão aplicados na prática nos capítulos seguintes.

Os pressupostos de segurança apresentados nesta seção estabelecem a base conceitual para a arquitetura proposta. A combinação dos princípios do modelo *Zero Trust*, a gestão de credenciais e segredos, e a proteção contra ataques como engenharia reversa e exploração de credenciais assegura um alto nível de resiliência contra ameaças modernas.

Além disso, a adoção de *IaC* permite que a segurança seja incorporada desde a concepção dos ambientes computacionais, minimizando riscos decorrentes de configurações inadequadas. A estratégia de defesa em profundidade aplicada neste modelo fortalece a proteção dos sistemas, dificultando a exploração de vulnerabilidades por agentes mal-intencionados. Esses pressupostos fundamentam a solução apresentada no capítulo 4, servindo como diretrizes para o desenvolvimento e avaliação da arquitetura proposta.

2.3.2 Superfície de ataque em autenticação não-interativa

A superfície de ataque é o conjunto de pontos dentro de um sistema que podem ser explorados por atacantes para comprometer a sua segurança. Em arquiteturas de autenticação não-interativa e gestão de segredos, a superfície de ataque pode ser ampliada devido a vulnerabilidades como:

- Armazenamento inadequado de credenciais em código-fonte, variáveis de ambiente ou arquivos de configuração.
- Exposição de chaves de API e *tokens* em *logs* ou repositórios públicos.
- Ataques de repetição, onde *tokens* válidos são reutilizados para acessar sistemas.
- Engenharia reversa, permitindo que atacantes extraiam segredos embutidos no código.
- Movimentação lateral, permitindo que um invasor comprometa múltiplos serviços ao explorar permissões excessivas.

A superfície de ataque está diretamente relacionada ao modelo de adversário, assumindo que um atacante pode explorar qualquer ponto exposto no sistema. No Capítulo 4, serão apresentados mecanismos para minimizar esses riscos.

2.3.3 Múltiplos fatores de autenticação

Segundo Oliveira *et al*. (2024), Múltiplos Fatores de Autenticação (MFA) são formas de se inserir mais um componente/fator no processo de autenticação, além do usuário e da senha. Pode-se, dessa forma, enviar mais um código por meio de mensagens SMS, gerar um código em um aplicativo de celular, enviar um código por e-mail, entre outros.

A MFA é um método de verificação de usuário que requer mais de um tipo de validação. Ela impede que atores mal-intencionados acessem uma conta, mesmo que tenham adquirido o nome do usuário e a senha. O MFA reduz a probabilidade de muitos

tipos de ciberataques. É comum que terceiros roubem nomes de usuários e senhas ou ataquem programaticamente contas de usuários. Um fator MFA adicional, como uma impressão digital ou senha de uso único, impede essas violações (Adaptado de Ometov *et al*., 2019).

Os padrões *Open* MFA são definidos no RFC 4226 (HOTP: Um algoritmo de senha única baseado em HMAC) e no RFC 6238 (TOTP: Algoritmo de senha única baseado em tempo). O PyOTP implementa suporte do lado do servidor para ambos os padrões. O suporte do lado do cliente pode ser habilitado enviando códigos de autenticação para usuários por SMS ou e-mail (HOTP) ou, para TOTP, instruindo os usuários a usar o *Google Authenticator, Authy* ou outro aplicativo compatível. Os usuários podem configurar *tokens* de autenticação em seus aplicativos facilmente usando a câmera do telefone para escanear os códigos QR otpauth:// fornecidos pelo PyOTP (Github/Pyauth, c2025).

No contexto da pesquisa, as técnicas do OTP e HOTP foram utilizadas na implementação do mecanismo de segurança para credenciais.

2.4 Gestão de identidades, segredos e cofre de senhas

O gerenciamento de acesso e identidade, conhecido como IAM, é uma disciplina de segurança que garante que as entidades corretas (pessoas ou dispositivos) possam acessar os recursos apropriados (aplicativos ou dados) quando necessário, sem restrições quanto aos dispositivos utilizados. O IAM consiste em sistemas e processos que permitem aos administradores de TI atribuir uma identidade digital única para cada entidade, autenticá-la no momento do login, autorizar seu acesso a recursos específicos e monitorar e gerenciar essas identidades ao longo de todo o seu ciclo de vida (IBM, 2024).

A gestão de identidades (*Identity Management* ou *GId*) ou a gestão de identidades e de acesso (*Identity and Access Management*) consiste em um conjunto de processos e tecnologias para gerenciar identidades de pessoas, serviços e coisas, bem como o relacionamento e a confiança entre essas. Esse recurso pode ser usado para garantir a

identidade de uma entidade e para prover procedimentos de autenticação, autorização, responsabilização e auditoria.

Diante da transformação digital acelerada, decorrente da pandemia da Covid-19, da constante evolução de tecnologias, das necessidades de usuários e de empresas por segurança, proteção de dados pessoais e usabilidade, a área de GId se mostra relevante e desperta interesse da academia, do governo e das empresas (RNP, 2022).

As soluções de IAM são integradas com uma variedade de tecnologias e ferramentas para ajudar a proteger a autenticação e a autorização em uma escala empresarial (Microsoft, c2025), entre elas:

a) SAML (*Security Assertion Markup Language*) – permite o logon único. Depois que um usuário foi autenticado com sucesso, o SAML notifica outros aplicativos que o usuário é uma entidade verificada. O SAML é importante, pois funciona em vários sistemas operacionais e computadores diferentes, o que permite garantir acesso de segurança em uma variedade de contextos.
b) OIDC (*OpenID Connect*) – adiciona um aspecto de identidade ao 0Auth 2.0, que é uma estrutura para autorização. Envia tokens que contêm informações sobre o usuário entre o provedor de identidade e o provedor de serviços. Esses tokens podem ser criptografados e conter informações sobre o usuário como o nome, endereço de e-mail, data de nascimento ou foto. Os tokens são fáceis de usar para serviços e aplicativos, o que torna o OIDC útil para autenticar jogos em celulares, mídias sociais e usuários de aplicativos.
c) SCIM (Sistema de Gerenciamento de Usuários entre Domínios) – auxilia as organizações a gerenciarem identidades de usuário de uma maneira padronizada, que funciona em vários aplicativos e soluções (provedores). Os provedores têm requisitos diferentes para informações de identidade de usuário e o SCIM permite criar uma identidade para um usuário em uma ferramenta que se integra com o provedor para que o usuário tenha acesso sem criar uma conta separada.

Outro tema de importância no contexto da pesquisa refere-se ao gerenciamento de segredos; segundo Red Hat (2024), a gestão de credenciais é um método para garantir a proteção das informações confidenciais necessárias para realizar operações diárias.

Reforça a segurança nos ambientes de desenvolvimento e produção da empresa sem depender de fluxos de trabalho de times *DevOps* ou das equipes de segurança.

O gerenciamento de segredos não é uma solução única, mas envolve a aplicação de várias medidas de segurança, como monitoramento e acesso autenticado, princípios de privilégio mínimo, controle de acesso baseado em funções, entre outras. Essas práticas garantem que os segredos sejam gerenciados corretamente, evitando a exposição de dados sensíveis a usuários não autenticados.

De acordo com Red Hat (2024), além de proteger os dados, a implementação do gerenciamento de segredos oferece diversas outras vantagens, entre elas:

a) **Prevenção da dispersão de segredos/ilhas de segurança:** ao centralizar o controle de segredos, é possível garantir que apenas um grupo pequeno da empresa tenha acesso especializado, que ninguém mais tem.

b) **Identificação de agentes maliciosos:** ao implementar requisitos de verificação sólidos, eliminam-se potenciais hackers com facilidade, impedindo suas tentativas de se passarem por usuários com credenciais.

c) **Automatização de credenciais:** ao automatizar a verificação e a admissão de credenciais, reduz-se a quantidade de trabalho manual necessário para autenticar usuários e permite-se que fluxos de trabalho continuem sem interrupções.

O terceiro tópico relacionado nesta seção, refere-se ao uso do cofre de senhas e sua principal funcionalidade nesse contexto. Segundo o Relatório de Investigações de Violações de Dados (DBIR) 2023, o roubo de senha é uma das três principais maneiras pelas quais invasores acessam virtualmente uma organização. Além disso, 95% das violações apresentam motivações financeiras.

De acordo com Kaspersky (c2025), especialistas em cibersegurança no mundo todo concordam que cofres de senhas são a maneira mais simples, fácil e segura de proteger senhas e dados confidenciais. Desde que a senha mestra seja "forte", suas informações confidenciais têm uma baixíssima chance de serem roubadas.

Segundo Hashicorp (2024), o Vault é um sistema de gerenciamento de segredos e criptografia baseado em identidade. Ele fornece serviços de criptografia que são

controlados por métodos de autenticação e autorização para garantir acesso seguro, auditável e restrito a segredos. É usado para proteger, armazenar e proteger segredos e outros dados confidenciais usando uma UI, CLI ou HTTP API.

O cofre de senhas Vault valida e autoriza os clientes antes de conceder acesso a segredos ou dados confidenciais armazenados. O Vault funciona principalmente com tokens e um token é associado à política do cliente. Cada política é baseada em caminho e as regras de política restringem as ações e a acessibilidade aos caminhos para cada cliente (Hashicorp, 2024). O fluxo de trabalho principal do Vault consiste em quatro etapas:

a) **Autenticar:** É o processo pelo qual um cliente fornece informações que o Vault usa para determinar se ele é quem diz ser. Uma vez que o cliente é autenticado em um método *auth*, um *token* é gerado e associado a uma política.
b) **Validar:** Faz a validação do cliente em relação a fontes confiáveis de terceiros, como Github, LDAP, *AppRole* e muito mais.
c) **Autorizar:** Um cliente é associado à política de segurança do Vault. Esta política é um conjunto de regras que definem a quais *endpoints* de API um cliente tem acesso com seu *token* do Vault. As políticas fornecem uma maneira declarativa de conceder ou proibir acesso a determinados caminhos e operações no cofre de senhas.
d) **Acessar:** Concede acesso a segredos, chaves e recursos de criptografia emitindo um *token* com base em políticas associadas à identidade do cliente. O cliente pode então usar seu *token* do cofre de senhas para operações futuras.

2.5 Métodos de ofuscação, criptografia e *hashing* (*hash* criptográfico)

No contexto da pesquisa é importante diferenciar os diversos métodos para proteção de dados. Entre eles, têm-se a ofuscação, a criptografia e o *hashing* criptográfico que serão detalhados nas seções seguintes.

Em um cenário digital em constante evolução, a segurança de aplicativos tornou-se fundamental para organizações que desejam proteger informações sensíveis e oferecer uma experiência segura aos usuários.

A criptografia e os *firewalls* são algumas das soluções comuns para diminuir a ameaça dos atacantes que tentam invadir o aplicativo. Porém, essas abordagens não ajudam a proteger o *software* quando o invasor é o próprio usuário final. Entre as várias técnicas disponíveis para proteger o código de diferentes ataques, a ofuscação de código é uma das alternativas mais populares para evitar a compreensão e adulteração do código-fonte (Behera; Bhaskari, 2015).

Nesse contexto, uma arquitetura adequada deve incorporar uma série de mecanismos de segurança que abordem aspectos críticos, garantindo proteção abrangente contra diferentes tipos de ameaças, desde a defesa contra acesso privilegiado até a prevenção de manipulação de credenciais.

2.5.1 Ofuscação

Técnicas de ofuscação são uma categoria geral de proteções de *software* amplamente adotadas para evitar adulteração maliciosa do código, tornando os aplicativos mais difíceis de entender e, portanto, mais difíceis de modificar. Técnicas de ofuscação são divididas em ofuscação de código e de dados, dependendo do ativo protegido (Viticchié *et al.,* 2016).

A ofuscação é a prática de tornar algo difícil de entender ou encontrar, a partir da alteração ou ocultação de sua aparência ou conteúdo. No contexto de cibersegurança e criptografia, a ofuscação se refere ao processo de tornar dados, códigos ou comunicações menos legíveis e mais difíceis de interpretar ou fazer engenharia reversa.

Com o intuito de fortalecer a segurança, a ofuscação é capaz de:

a) Ocultar informações confidenciais de acesso não autorizado ou uso indevido.
b) Proteger propriedade intelectual (como algoritmos e códigos proprietários).

c) Impedir engenharia reversa, adulteração ou análise de código ou estruturas de dados.
d) Complementar outras medidas de segurança, como criptografia, autenticação e controle de acesso, não devendo ser considerada como uma única linha de defesa.

A ofuscação torna os ataques mais complexos, mas não pode bloqueá-los completamente (Khairunisa *et al*., 2021). A ofuscação de dados visa ocultar tanto o conteúdo variável quanto o uso, e pode ser aplicada, por exemplo, a dados críticos, como: IDs de usuário, contadores, datas de expiração ou dados sensíveis à privacidade, como dados médicos (Viticchié *et al.,* 2016).

Segundo a pesquisa de Viticchié *et al.* (2016), a ofuscação de dados afeta significativamente o tempo para concluir uma tarefa de ataque e a eficiência do ataque, embora não afete a exatidão do seu resultado. Observa-se que a presença de ofuscação força o invasor a recorrer a mais ferramentas disponíveis para desofuscar o código.

Com isso, a ofuscação pode afetar o desempenho e a capacidade de manutenção, pois a complexidade e a sobrecarga adicionais podem tornar o código mais lento para executar e mais difícil de manter ou atualizar.

Dessa forma, a ofuscação pode ser integrada como um componente de uma estratégia de segurança abrangente, que inclua autenticação, controle de acesso e modelos baseados em *Zero Trust*. Embora seja uma ferramenta útil para fortalecer a postura de segurança de um sistema, ela não deve ser vista como a única medida de proteção.

A ofuscação de credenciais, conforme a proposta neste trabalho, atua no contexto em que uma API (Interface de Programação de Aplicações) não é, em si, uma “solução”, mas uma ponte de comunicação que permite a interação entre diferentes *softwares* ou sistemas.

A ofuscação de código transforma o programa original em uma versão semanticamente equivalente, mas mais difícil de ser compreendida por invasores. No entanto, nenhuma técnica atual de ofuscação é completamente eficaz contra ataques de engenharia reversa. Pesquisadores e indústrias continuam desenvolvendo e aplicando

novas abordagens para melhorar a proteção, embora o código ainda possa ser vulnerável a violações (Behera; Bhaskari, 2015).

Entretanto, de acordo com Micucci (2024), a ofuscação de código consolidou-se como uma estratégia fundamental para o fortalecimento da segurança de *software*, proporcionando proteção à propriedade intelectual e garantindo a integridade, a confidencialidade e a disponibilidade das informações em ambientes digitais. Embora não elimine totalmente o risco de comprometimento, a ofuscação dificulta significativamente as tentativas de invasão e exploração de sistemas. Assim, a adoção de técnicas de ofuscação, com aplicação correta do seu funcionamento, assegura a proteção dos sistemas e minimiza vulnerabilidades.

Por fim, na prática, técnicas como criptografia, proteção pelo lado do servidor, soluções de segurança baseadas em *hardware*, códigos nativos assinados, provas de adulteração, marcas d'água, envelhecimento do *software* e empacotamento estão entre os métodos mais utilizados para dificultar ou evitar a detecção e manipulação por invasores.

No entanto, é fundamental que o provedor avalie por quanto tempo a ofuscação será eficaz, ou seja, o tempo necessário para que um invasor consiga compreender o código. Com base nisso, outras estratégias de ofuscação podem ser aplicadas ao código original, dificultando ainda mais o acesso ao algoritmo ou à lógica do sistema (Adaptado de Behera; Bhaskari, 2015).

2.5.2 Criptografia

A criptografia é um método de segurança que transforma dados legíveis (texto plano) em um formato ilegível (texto cifrado) usando um algoritmo e uma chave secreta. O processo é reversível, ou seja, é possível recuperar os dados originais por meio de descriptografia, desde que se tenha a chave correta. O objetivo da criptografia é proteger a confidencialidade das informações, garantindo que apenas as pessoas autorizadas possam acessar os dados.

A ofuscação é diferente da criptografia em muitos aspectos. Em primeiro lugar ela não exige nenhuma transformação inversa. Em seguida, não é necessário que um invasor procure o código original o tempo todo, pois o ataque pode ser bem-sucedido sem ter o código-fonte original. E, por fim, o texto cifrado não terá valor sem a chave, pois um programa ofuscado pode ser executado sem nenhuma informação adicional (Behera; Bhaskari, 2015).

No processo de ofuscação, quando combinado com outras camadas de segurança, como a arquitetura de *Zero Trust*, surge um componente de segurança valioso. Ao contrário da criptografia, que depende de um armazenamento seguro da chave secreta, a ofuscação utilizada neste contexto reduz esse risco, tornando-se uma abordagem eficiente e robusta dentro da arquitetura proposta. Essa eficácia é evidenciada ao longo desta pesquisa.

Segundo Maziero (2019), um dos principais problemas no uso da criptografia simétrica para a criação de um canal de comunicação segura é a troca de chaves, ou seja, o estabelecimento de um segredo comum entre os interlocutores. Caso eles não estejam fisicamente próximos, criar uma senha secreta comum, ou substituir uma senha comprometida, pode ser um processo complicado e demorado.

Um resumo criptográfico (*cryptographic hash*) (Menezes *et al*., 1996 *apud* Maziero, 2019) é uma função y = hash(x) que gera uma sequência de *bytes* y de tamanho pequeno e fixo (algumas dezenas ou centenas de *bytes*) a partir de um conjunto de dados x de tamanho variável aplicado como entrada. Os resumos criptográficos são frequentemente usados para identificar unicamente um arquivo ou outra informação digital, ou para atestar a sua integridade: caso o conteúdo de um documento digital seja modificado, seu resumo também será alterado.

Segundo Brito (2019), criptografar é um pouco melhor que guardar o *Password encoded*. Criptografia é uma via de mão dupla, onde algo criptografado é possível de ser descriptografado a partir de uma chave. E esse é o problema fundamental de criptografar senhas. Onde estará a chave? O processo de validação da senha precisa descriptografar para comparar; se um atacante comprometer a chave terá acesso a todas as senhas do banco.

2.5.3 *Hashing* (*hash* criptográfico)

Como visto na seção anterior, os algoritmos de resumo criptográfico mais conhecidos e utilizados atualmente são os da família SHA (*Secure Hash Algorithms*). O *hashing* é um processo criptográfico que converte dados de entrada de qualquer tamanho em uma sequência de caracteres de tamanho fixo, normalmente um número hexadecimal. A saída de dados, chamada de valor *hash* ou resumo, é exclusiva dos dados de entrada e serve como uma impressão digital.

Ao contrário da criptografia, o *hashing* é um processo unidirecional, o que significa que é computacionalmente inviável reverter o *hash* para obter os dados originais. Na cibersegurança, o *hashing* é amplamente usado para armazenamento de senhas, verificação de integridade de dados e assinaturas digitais (Roadmap.sh, 2024).

Em uma adaptação de Roadmap.sh (2024), algoritmos de *hash* comuns incluem MD5, SHA-256 e *bcrypt*. O *hashing* ajuda a detectar alterações não autorizadas nos dados, pois mesmo uma pequena alteração na entrada produz um valor hash significativamente diferente. No entanto, a complexidade de uma função *hash* é crucial, pois algoritmos fracos podem ser vulneráveis a ataques de colisão; neles, entradas diferentes produzem o mesmo hash, potencialmente comprometendo medidas de segurança que dependem da exclusividade dos valores *hash.*

O uso de algoritmos de *hash* criptográfico garante a integridade e a segurança das informações em sistemas computacionais. Entre os algoritmos mais utilizados estão o SHA-256 e o HMAC (*Hash-based Message Authentication Code*), ambos amplamente aplicados em diversas soluções de segurança digital. No entanto, existem diferenças significativas entre o uso de um *hash* puro como o SHA-256 e um HMAC, envolvendo o contexto de autenticidade e integridade de dados.

A utilização conjunta do HMAC e do SHA-256 para fins de autenticação e proteção de credenciais representa uma combinação viável e eficaz, em sistemas que demandam um alto nível de segurança. Essa abordagem não apenas garante a integridade e autenticidade das mensagens, mas também oferece proteção adicional contra ataques de terceiros, como ataques de força bruta e de interceptação de dados (Schneier, 2015).

O SHA-256 é amplamente utilizado para gerar *hashes* seguros, garantindo que qualquer alteração nos dados seja facilmente detectada. No entanto, como discutido por Preneel (2019), o uso de um *hash* puro não é suficiente para garantir a autenticidade das informações. Nesse contexto, o HMAC se torna uma solução complementar interessante, pois utiliza uma chave secreta (semente) em combinação com a função de *hash*, fornecendo uma camada adicional de segurança e verificando se a mensagem foi gerada por uma fonte confiável.

Quando aplicado ao processo de proteção de credenciais, o uso conjunto de HMAC e SHA-256 se torna ainda mais relevante. Conforme Cao *et al*. (2024), a proteção de credenciais por meio de técnicas de ofuscação é uma prática recomendada para dificultar a exposição de dados sensíveis. A ofuscação de credenciais adiciona uma camada de complexidade, tornando mais difícil para um atacante interpretar os dados, mesmo que consiga acessá-los.

A combinação do HMAC, SHA-256 e técnicas de ofuscação cria um ambiente onde a autenticidade e confidencialidade das informações são asseguradas. Segundo Krawczyk, Bellare e Canetti (1997), o uso de HMAC protege contra ataques de retransmissão (*replay attacks*) e garante que apenas as partes que possuem a chave secreta possam validar a autenticidade das mensagens. Isso torna o uso conjunto de HMAC e SHA-256 uma prática recomendada em aplicações que exigem alto nível de segurança, como em sistemas de autenticação multifatorial (MFA) e em soluções de gestão de segredos.

Portanto, a combinação de SHA-256 e HMAC, aliada a técnicas de ofuscação de credenciais, constitui uma estratégia eficaz para proteger informações sensíveis contra acessos não autorizados. Essa abordagem reduz os riscos de exploração de vulnerabilidades, protege contra ataques de interceptação e melhora a segurança geral dos sistemas digitais.

2.5.4 Discussão entre o uso da ofuscação e criptografia dentro da arquitetura

No contexto de autenticação de APIs em processos não-interativos, a escolha entre ofuscação e criptografia levanta questões relevantes sobre segurança e eficiência. Embora a criptografia seja amplamente utilizada para proteger dados sensíveis, a ofuscação se apresenta como uma solução complementar e eficiente em cenários específicos, especialmente quando o objetivo é dificultar a engenharia reversa e proteger a propriedade intelectual dos sistemas.

Segundo Schneier (2015), a criptografia transforma dados legíveis em formatos ilegíveis para garantir confidencialidade. Contudo, em sistemas não-interativos, sua implementação pode aumentar a complexidade, exigindo um rigoroso gerenciamento de chaves sem necessariamente impedir ataques de engenharia reversa. Já a ofuscação, embora não impeça o acesso aos dados, dificulta a análise do código-fonte, tornando mais desafiador para um atacante compreender a lógica de autenticação e identificar credenciais (Cao *et al*., 2024).

Preneel (2019) aponta que a ofuscação é particularmente eficaz para proteger aplicações contra ameaças internas e externas, especialmente em ambientes onde o código pode ser acessado por terceiros. Em sistemas de autenticação de APIs, a ofuscação protege credenciais e dificulta a compreensão da lógica de autenticação, mesmo que o código seja exposto. Essa característica é essencial em cenários não-interativos, onde não há intervenção humana para validar requisições.

A combinação de ofuscação com HMAC e funções de *hash*, como SHA-256, adiciona uma camada extra de proteção. Conforme Krawczyk, Bellare e Canetti (1997), essa abordagem dificulta o acesso a credenciais mesmo em casos de comprometimento parcial do sistema, reduzindo o risco de exploração de vulnerabilidades.

A ofuscação alinha-se aos princípios do modelo *Zero Trust*, que parte do pressuposto de que nenhum dispositivo ou usuário é confiável por padrão (Kindervag, 2010). Esse modelo exige verificações contínuas de identidade e controles rigorosos de acesso. Nesse contexto, a ofuscação complementa a segurança, protegendo credenciais e

lógicas de autenticação contra ataques internos e externos, mesmo em sistemas parcialmente comprometidos.

Cao *et al*. (2024) reforçam que, em APIs abertas ou sistemas onde a exposição do código é provável, a ofuscação se torna uma camada adicional eficaz contra ataques baseados em engenharia reversa, reduzindo significativamente as chances de exploração de vulnerabilidades.

Embora a criptografia continue sendo essencial para garantir a confidencialidade de dados em trânsito, ela pode impactar negativamente o desempenho de sistemas que exigem autenticações frequentes e em tempo real. A execução constante de algoritmos criptográficos robustos pode gerar sobrecarga nos servidores, aumentando a latência e prejudicando a experiência do usuário (Krawczyk; Bellare; Canetti, 1997).

Por outro lado, a ofuscação oferece uma alternativa mais leve, dispensando a necessidade de chaves ou processos complexos de descriptografia, reduzindo assim a sobrecarga computacional sem comprometer a segurança.

Em sistemas que realizam várias chamadas por segundo, como APIs de alta frequência, a substituição parcial da criptografia pela ofuscação pode melhorar o desempenho sem sacrificar a proteção. Ferramentas de monitoramento ajudam a identificar processos que consomem mais recursos computacionais, permitindo a avaliação precisa de quais elementos podem ser otimizados por meio da ofuscação (Preneel, 2019).

Segundo Collberg *et al.* (1997), é possível combinar mais de uma técnica de ofuscação, de modo que se possa obter mais robustez, de acordo com as seguintes observações:

a) O *software* aplicativo ofuscado nunca passa por uma interrupção devido às limitações de rede. Além disso, ele não precisa de um *hardware* para criptografar ou descriptografar o código. Portanto, não há necessidade de assinatura digital para que o aplicativo de *software* autentique se o aplicativo tem código seguro de fonte confiável ou não.

b) As técnicas de criptografia precisam de algum *hardware* específico para agir com eficácia, com o código assinado para colocar alguma restrição, que não

deve depender da plataforma de *hardware*. Esse uso de *hardware* dedicado também aumenta o custo para o comprador do *software*.

c) A descompilação será mais difícil para o código ofuscado, mesmo depois de despender tempo e esforço suficientes, mas isso é possível com os algoritmos e as estruturas de dados. Portanto, a principal intenção é aumentar o tempo e o esforço, de modo que seja eficientemente inviável para um adversário fazer a engenharia reversa do código ou do *software* ofuscado.

d) Por meio da ofuscação, é possível emaranhar o código e eliminar a maioria dos *links* lógicos, de modo que o código transformado se torna complexo o suficiente para análise e modificações não autorizadas.

e) Há muitos algoritmos de criptografia complexos que não são possíveis de implementar devido às limitações de memória e de largura de banda da rede.

f) Nos progressos recentes, a teoria da ofuscação ainda precisa de uma avaliação da qualidade da prática, ofuscando as transformações de maneira mais rápida e fácil.

Por fim, a escolha entre ofuscação e criptografia depende do contexto e das necessidades específicas do sistema. Enquanto a criptografia é indispensável para proteger dados sensíveis em trânsito, a ofuscação se destaca como uma solução eficiente para proteger credenciais e lógica de autenticação em sistemas de API não-interativos, especialmente em arquiteturas baseadas no modelo *Zero Trust*. A combinação dessas técnicas, ajustada às características do sistema, fortalece a segurança, reduz vulnerabilidades e garante um equilíbrio entre proteção e desempenho.

2.6 Modelagem de ameaças, adversário e teste de invasão (*pentest*)

Como ameaça, pode ser considerada qualquer ação que coloque em risco as propriedades de segurança do sistema. Alguns exemplos de ameaças às propriedades básicas de segurança seriam: ameaças à confidencialidade, ameaças à integridade e ameaças à disponibilidade. As ameaças podem ou não se concretizar, dependendo da

existência e da correção dos mecanismos construídos para evitá-las ou impedi-las. As ameaças podem se tornar realidade à medida em que existam vulnerabilidades que permitam sua ocorrência (Maziero, 2019).

A modelagem de ameaças é uma abordagem estruturada utilizada para identificar vulnerabilidades de segurança em um aplicativo e seu ambiente. De acordo com a Cloudflare (c2025a), esse processo envolve a criação de uma representação detalhada do sistema, incluindo seus componentes e fluxos de dados, para identificar possíveis pontos fracos.

Desenvolvedores e engenheiros de segurança devem aplicar essa prática ao longo de todo o ciclo de desenvolvimento de *software* (SDLC), priorizando a identificação e mitigação de vulnerabilidades antes de a aplicação ser colocada em produção. Embora não torne os sistemas completamente invulneráveis, a modelagem de ameaças contribui significativamente para aumentar a sua resiliência (Cloudflare, c2025a).

O processo de modelagem geralmente segue quatro etapas principais:

a) **Avaliação e representação:** Análise do sistema para identificar componentes críticos e fluxos de dados.
b) **Identificação de vulnerabilidades:** Mapeamento de possíveis falhas de segurança.
c) **Correção:** Implementação de alterações para mitigar as falhas identificadas.
d) **Validação:** Verificação da eficácia das correções aplicadas e mitigação das ameaças.

O registro de cada etapa resulta em um documento que se torna a base do modelo de ameaças para o aplicativo, promovendo a segurança contínua ao longo do ciclo de vida do *software* (Conklin, [2024?]). Essa análise considera o sistema do ponto de vista de um potencial atacante, permitindo prever cenários adversos e aumentar a eficiência das defesas.

Organizações podem adotar diversas metodologias para estruturar a modelagem de ameaças, dependendo da complexidade do sistema e das prioridades de segurança. Entre as mais conhecidas estão:

a) **STRIDE:** Desenvolvida pela Microsoft, abrange seis tipos de ameaças: falsificação (*spoofing*), adulteração (*tampering*), repúdio (*repudiation*),

divulgação de informações (*information disclosure*), negação de serviço (*denial of service*) e elevação de privilégio (*elevation of privilege*).

b) **PASTA:** Processo estruturado em sete etapas para simulação de ataques e análise de ameaças.
c) **VAST:** Focado em agilidade e simplicidade, utiliza visualizações para facilitar a identificação de vulnerabilidades.
d) **SQUARE:** Enfatiza a engenharia de requisitos de segurança no início do desenvolvimento, ajudando a identificar preocupações antes que elas se tornem problemas.

Cada metodologia oferece ferramentas e diretrizes específicas para abordar ameaças de forma sistemática, permitindo que as organizações escolham a mais adequada às suas necessidades.

O *Mitre Att&ck* é uma base de conhecimento amplamente reconhecida que descreve táticas e técnicas adversárias com base em observações reais. Ele serve como referência para criar modelos de ameaças e metodologias específicas em diferentes setores, ajudando organizações a entender o comportamento de atacantes e a antecipar as suas ações (Mitre *Att&ck*, 2024).

Nesta pesquisa, o modelo de adversário do *Mitre Att&ck* foi utilizado para identificar vetores de ataque relevantes, avaliar vulnerabilidades críticas e mapear estratégias de mitigação, alinhando-se às boas práticas de segurança e fortalecendo a arquitetura proposta. A integração dessa abordagem permitiu uma visão detalhada do comportamento dos adversários e serviu como base para os testes de invasão realizados.

Os testes de invasão simulam ataques reais para identificar vulnerabilidades e avaliar a eficácia das defesas. *Hackers* éticos utilizam as mesmas ferramentas, técnicas e procedimentos que invasores reais, permitindo que as organizações descubram falhas antes que sejam exploradas (Cloudflare, c2025b). Além de identificar vulnerabilidades, o *pentest* avalia a resposta a incidentes e a resiliência geral dos sistemas, fornecendo insights práticos para a correção de falhas.

De acordo com Check Point (c1994-2025b), o *pentest* é essencial para melhorar a segurança organizacional, corrigindo vulnerabilidades em sistemas e APIs, fortalecendo a gestão de incidentes e aprimorando processos de cibersegurança. Essa prática, em

conjunto com a modelagem de ameaças e o uso do *Mitre Att&ck*, oferece uma abordagem abrangente para mitigar riscos, antecipar cenários adversos e proteger os sistemas contra ataques sofisticados.

2.7 Segurança por perímetro, segmentação de rede, movimento lateral e *VPN*s

Tradicionalmente, redes de computadores eram protegidas por uma abordagem de segurança baseada no perímetro, onde *firewalls* e dispositivos de segurança controlavam o tráfego que entrava e saía da rede. Embora eficaz por décadas em ambientes corporativos, essa abordagem apresenta limitações significativas no contexto atual de redes dinâmicas e distribuídas.

Uma preocupação central é o movimento lateral, no qual um agente mal-intencionado, após comprometer um sistema dentro do perímetro, pode se deslocar entre servidores e aplicativos, explorando vulnerabilidades internas.

A segurança do perímetro visa proteger os limites de uma rede contra acessos não autorizados, empregando medidas como *firewalls*, sistemas de detecção de intrusão (IDS), e controles físicos e lógicos (Twingate, 2024). Entre os principais componentes dessa abordagem destacam-se:

a) ***Firewalls*:** Monitoram e controlam o tráfego de entrada e saída com base em políticas de segurança.
b) **Controles de acesso:** Incluem autenticação por senhas, biometria e cartões inteligentes.
c) **Análise de vídeo:** Detecta e responde a atividades suspeitas em pontos de entrada e saída.
d) **Sistemas de detecção de intrusão (IDS):** Monitoram a rede em busca de atividades maliciosas.

Apesar dessas medidas, o modelo de segurança por perímetro tem limitações intrínsecas. Por exemplo, o uso de *VPNs* possibilita conexões seguras com criptografia,

mas não resolve o problema da movimentação lateral. Uma vez dentro da rede, invasores ou usuários mal-intencionados podem explorar recursos e comprometer outros sistemas.

Além disso, a falta de visibilidade em segmentos de rede e a ausência de controles adequados agravam os riscos, especialmente em ambientes legados com configurações inadequadas ou vulnerabilidades conhecidas (CVEs) (Twingate, 2024).

Para superar as limitações da segurança por perímetro, é necessário adotar estratégias mais granulares, como a microssegmentação. Essa abordagem vai além do perímetro, isolando cada aplicação ou serviço em segmentos específicos e proporcionando controle detalhado sobre o tráfego interno, também conhecido como tráfego "Leste-Oeste".

A microssegmentação permite:

a) **Isolamento efetivo:** Aplicações e ambientes são segmentados no nível binário, possibilitando políticas de segurança específicas para cada contexto.
b) **Redução da superfície de ataque:** Restringe a movimentação lateral e limita o impacto de invasões.
c) **Proteção robusta:** Isolamento de *containers* e dispositivos aumenta a resiliência contra ataques.

Diferente dos métodos tradicionais, que dependem amplamente de *firewalls* e segmentação por IPs, a microssegmentação associa políticas de segurança diretamente ao comportamento das aplicações. Isso garante que, mesmo que um invasor comprometa um sistema, encontrará barreiras significativas para explorar outros recursos da rede.

Enquanto a segurança por perímetro foca na proteção das bordas da rede contra ameaças externas, a microssegmentação aprofunda o controle interno, assegurando que cada aplicação, dispositivo ou *contêiner* possua políticas específicas. Segundo Macy (2023) e Twingate (2024), essa abordagem supera as limitações do modelo tradicional ao oferecer maior flexibilidade e adaptabilidade para proteger redes modernas.

A combinação da microssegmentação com controle de acesso granular e validação contínua, permite enfrentar os desafios de redes distribuídas e ambientes dinâmicos. Ao reduzir a movimentação lateral e isolar aplicações, a microssegmentação complementa

outras estratégias de segurança, como o modelo *Zero Trust*, criando uma arquitetura resiliente contra ataques sofisticados.

Capítulo 3

Trabalhos Relacionados

A literatura apresenta a necessidade de estudos relacionados a falhas de segurança para *IaC*, conforme identificado no estudo de mapeamento sistemático conduzido por Rahman *et al*. (2019). Entre os temas correlatos à segurança da informação, destacam-se abordagens como *Security Smells* (Rahman *et al*., 2021a), práticas de *DevSecOps* (Ibrahim; Yousef; Medhat, 2022) e estratégias de gerenciamento de segredos em *IaC* (Rahman *et al*., 2021b). Esses trabalhos fornecem *insights* importantes, mas ainda há lacunas na aplicação de conceitos práticos como o *Zero Trust* à gestão de segredos e segurança em ambientes *IaC*.

Um estudo recente da Codacy (2023) destaca a crescente relevância da gestão segura de credenciais em sistemas modernos. Aplicativos atuais dependem de segredos para integrações com APIs, serviços de terceiros e bancos de dados em nuvem, ampliando a quantidade de credenciais que precisam ser protegidas. Essa dependência intensifica os desafios de gerenciamento, especialmente em ambientes distribuídos, onde erros podem levar a exposições críticas.

Apesar da importância crescente do tema, poucos trabalhos se dedicam a investigar mecanismos robustos para proteger segredos e mitigar o roubo de credenciais em sistemas automatizados.

Dado esse cenário, este capítulo explora os trabalhos relacionados à presente pesquisa, contextualizando as contribuições existentes e identificando como a combinação de práticas de segurança com *IaC* e o modelo *Zero Trust* pode avançar no enfrentamento dos desafios associados à gestão de segredos. A análise inclui abordagens inovadoras que buscam alinhar a segurança a princípios modernos de automação e resiliência, criando um alicerce para a pesquisa em desenvolvimento.

3.1 Infraestrutura como código, análise estática e segurança em *IaC*

Rahman *et al*. (2019) investigaram os desafios da pesquisa em *IaC* por meio de uma revisão sistemática da literatura (RSL). O objetivo principal foi identificar as diversas áreas de pesquisa que envolvem o campo da *IaC*. Os quatro principais tópicos identificados são (a) estrutura/ferramenta para *IaC*; (b) uso de *IaC*; (c) estudos empíricos relacionados à *IaC*; e (d) testes em *IaC*.

Os autores concluíram que, embora existam vários estudos sobre estruturas e ferramentas, a investigação no contexto de defeitos de *IaC* e falhas de segurança ainda está em uma fase inicial. Os resultados do SLR estão perfeitamente alinhados com a investigação proposta neste artigo: de fato, o estado da investigação e da prática em *IaC* ainda é imaturo, o que exige estudos mais empíricos e centrados na indústria, como o realizado ao longo deste artigo.

Também em Rahman *et al*. (2021a), os autores apresentaram um catálogo de sete práticas inadequadas: *Security Smells* em scripts *IaC*. Estes foram extraídos da análise qualitativa de *scripts* Puppet em repositórios de código aberto. As falhas identificadas nos códigos incluem: (1) concessão de privilégios de administrador por padrão, (2) senhas vazias, (3) segredos codificados, (4) ligação de endereço IP inválido, (5) comentários suspeitos (como ‘TODO’ ou ‘FIXME ‘), (6) uso de HTTP sem TLS e (7) uso de algoritmos de criptografia fracos. No entanto, isso é novamente limitado aos *scripts* Puppet e nem todas as falhas nos códigos podem ser generalizadas para outras linguagens ou ferramentas.

A análise de configuração estática envolve a avaliação de scripts de configuração para detectar códigos e falhas de segurança sem realmente executar os *scripts*. Isto permite que os especialistas da *IaC* identifiquem e resolvam os riscos antes da implantação. O processo pode ser realizado manualmente ou com o uso de ferramentas automatizadas que examinam os scripts em busca de uma categoria específica de falhas nos códigos.

Ainda sobre a verificação estática em *IaC*, Konala, Kumar e Bainbridge (2023), em um recente artigo, abordam uma análise de configuração estática em scripts de *IaC*.

O estudo aponta que as ferramentas de *IaC* podem economizar tempo em comparação com as configurações manuais, mas apresentam limitações. Uma das mais importantes é a dificuldade em identificar e analisar códigos inadequados, especialmente no que se refere a riscos de segurança.

Por conta disso, profissionais que trabalham com *IaC* precisam adotar recursos extras para realizar verificações manuais personalizadas durante o desenvolvimento ou recorrer a ferramentas especializadas. Caso esses códigos inadequados não sejam detectados, o risco de ciberataques pode aumentar significativamente.

Para resolver os problemas de defeitos no *IaC*, métodos semelhantes aos utilizados no desenvolvimento de *software* podem ser implementados na forma de análise de configuração estática e de configuração dinâmica. Um dos métodos mais comuns e eficientes em termos de recursos é a análise de configuração estática por *scripts IaC*. Este tipo de análise concentra-se na sintaxe do *IaC*.

É vantajoso para as organizações, pois não requer implantação de infraestrutura, economizando custos e tempo para a criação de ambientes de teste isolados (Konala; Kumar; Bainbridge, 2023).

Os autores Konala, Kumar e Bainbridge (2023) apresentam um exame detalhado dos mais recentes avanços no campo da análise de configuração estática em *IaC,* através de uma extensa revisão da literatura, reunindo informações sobre as diferentes técnicas e ferramentas desenvolvidas para este fim, tanto acadêmica quanto comercialmente. Além disso, destacam as áreas que requerem investigação adicional para aumentar a eficácia e eficiência da detecção de defeitos de configuração em *IaC*.

Concluindo o estudo, os mencionados autores relatam que a maioria das ferramentas e técnicas são desenvolvidas para provisionamento de infraestrutura, em vez de gerenciamento de configuração e construção de imagens. Isto levanta preocupações, visto que configurar *software* é uma tarefa de alto risco, com agentes mal-intencionados constantemente visando sistemas de *software*. Portanto, é crucial que os pesquisadores desenvolvam técnicas eficientes e avançadas para detectar defeitos no gerenciamento de configuração e na construção de imagens.

Ibrahim, Yousef e Medhat (2022) propõem um módulo *DevSecOps* para *IaC*, totalmente seguro e automatizado. Os autores coletaram e organizaram todas as etapas de segurança encontradas em trabalhos anteriores, que serviram de base para a criação deste módulo. As ferramentas utilizadas foram o Terraform para provisionamento e o Ansible para configuração da infraestrutura.

Na sua pesquisa, Ibrahim, Yousef e Medhat (2022) abordam uma pergunta importante: "como podemos implementar um módulo para alcançar *DevSecOps* em *IaC*?" Ou seja, uma tentativa de incluir práticas seguras de codificação em um ambiente *DevOps*, utilizando a *IaC* para a sua implementação.

Segundo Ibrahim, Yousef e Medhat (2022), foi possível resolver a questão da segurança na *IaC* aplicando o módulo proposto no estudo. O trabalho recomenda o módulo DevSecOps para a *IaC*, que automatiza e protege todo o processo, dentro da cadeia SDLC. O módulo foi avaliado e testado em duas frentes, o que permitiu validar a sua eficiência.

Em suma, o trabalho abordou uma das muitas possibilidades para tentar garantir segurança na *IaC*, ao aplicar práticas de desenvolvimento, operação e segurança combinadas.

Rahman e Williams (2021), tratando sobre diferentes tipos de *security smells* em *IaC*, afirmam que os padrões de configurações que apontam para falhas de segurança de scripts *IaC* são predominantes − e o mais comum é o segredo codificado. O segredo codificado considera três tipos de segredos: senhas, nomes de usuários e chaves criptográficas.

Segundo Rahman e Williams (2021), padrões de configurações que apontam para falhas de segurança bem conhecidas aparecem para linguagens existentes − como Java − e também para *IaC*. Uma possível explicação atribui a prevalência de segredos codificados à natureza do desenvolvimento de *IaC,* ou seja, se os profissionais configuram contas de usuário, nomes de usuário e senhas, estes provavelmente serão especificados em scripts *IaC*.

Outra possível explicação pode dizer respeito à falta de disponibilidade e uso de ferramentas de credenciais, pois talvez os profissionais não tenham ferramentas

adequadas para gerenciar senhas e nomes de usuários em scripts *IaC*. E podem não estar cientes da existência de ferramentas específicas para gerenciamento de segredos, como o Hashicorp Vault.

Por fim, Rahman e Williams (2021) relatam que a falta de conscientização sobre cibersegurança pode ser outra possível explicação. Os profissionais que desenvolvem *scripts IaC* podem não saber sobre as consequências dos padrões de configurações que apontam para falhas de segurança e podem ter contribuído para a prevalência de tais problemas.

3.2 *Zero Trust*

Dentro da abordagem de *Zero Trust* e *IaC,* Rong *et al*. (2022) apresentam uma estrutura *OpenIaC,* estabelecendo um corpo de padrões de interoperabilidade dentro de um ecossistema de rede e serviços 5G bloqueados por fornecedores, o que compromete o compartilhamento e a mobilidade. A ideia dentro do contexto da pesquisa conduz a uma possível abordagem do *Zero Trust* e *IaC*.

Em resumo, segundo Rong *et al.* (2022), a proposta permite fornecer acesso móvel e sem fronteiras a serviços de infraestrutura, sendo que os usuários podem acessar serviços a qualquer hora e em qualquer lugar em qualquer dispositivo, o que aumenta a conveniência e a produtividade, mas os riscos de segurança aumentam inevitavelmente. Nesse sentido, o *OpenIaC* propõe uma solução de segurança *Zero Trust* usando contratos inteligentes e identidade descentralizada.

A pesquisa do *OpenIaC* permitiu implementar todos os recursos de computação necessários − incluindo o armazenamento, configuração de rede e segurança de forma programável −, e aproveitando todos os recursos e vantagens da *IaC* (Rong *et al*., 2022).

O estudo realizado por Ahmadi (2024) permitiu analisar os desafios e oportunidades na implementação de uma arquitetura *Zero Trust* (ZTA) na nuvem, abordando os temas relacionados ao contexto e revelando o impacto da ZTA na mitigação do movimento lateral, na redução da probabilidade de ameaças internas, no

aprimoramento da microssegmentação da rede e na melhoria do gerenciamento de identidade e acesso. A análise comparativa demonstra melhorias significativas em incidentes de segurança após a implementação da ZTA. Além disso, o estudo destaca as melhores práticas para a adoção da ZTA e descreve avanços futuros, incluindo a sua integração com tecnologias emergentes, como aprendizado de máquina e inteligência artificial.

Na sua pesquisa, Ahmadi (2024) avaliou qualitativamente aspectos importantes relacionados à segurança, que são o movimento lateral, a avaliação das ameaças e superfície de ataque, o gerenciamento de identidade e a microssegmentação.

Na análise, o autor pôde verificar as melhores práticas a serem consideradas para a implementação de uma ZTA e identificar estratégias de princípios fundamentais que sustentam a sua implementação bem-sucedida, incluindo acesso de privilégio mínimo, microssegmentação e técnicas de criptografia. A pesquisa ressalta o papel fundamental da ZTA no fortalecimento da segurança da rede em nuvem e oferece *insights* valiosos para profissionais e pesquisadores.

No artigo que estabelece estratégia de confiança zero, Mehraj e Banday (2020) traçam uma estratégia conceitual para a implantação do modelo *Zero Trust.* O crescimento no uso de serviços em nuvem trouxe consigo diversos desafios relacionados à segurança e privacidade, incluindo roubo de identidade, violações de dados, e questões envolvendo integridade e confidencialidade das informações. Esses fatores tornaram o gerenciamento da confiança um dos aspectos mais complexos da computação em nuvem.

Segundo Mehraj e Banday (2020), o modelo *Zero Trust* é uma abordagem moderna para cibersegurança que assume que nada, dentro ou fora da rede, é confiável. Ao aplicar esse conceito na computação em nuvem, garante-se segurança baseada em identidade e localização, além de possibilitar a inspeção de tráfego e o monitoramento contínuo de dispositivos e usuários conectados, reduzindo a complexidade do ambiente de TI.

Por fim, o artigo aborda os princípios fundamentais da abordagem *Zero Trust*, detalhando cada um deles e explorando sua aplicabilidade na implementação de uma estratégia de adoção de confiança zero.

3.3 Gestão de segredos, roubo de credenciais e ofuscação

Em Rahman *et al*. (2019), os autores apresentam um estudo com doze práticas para gerenciamento de segredos em *IaC*. O objetivo do artigo é ajudar os profissionais no desenvolvimento seguro de scripts de *IaC,* identificando práticas para gerenciamento de segredos. A prevalência de segredos em *scripts IaC* necessita da integração de gerenciamento adequado no desenvolvimento de *IaC*. Essa integração, no entanto, pode ser desafiadora devido à falta de práticas essenciais para gerenciar segredos.

Rahman *et al*. (2019) afirmam que a falta de práticas relacionadas ao gerenciamento de segredos para *IaC* pode limitar o uso das ferramentas como Ansible, Chef e Puppet. Segundo os autores, os *scripts IaC* automatizam o processo de provisionamento de infraestrutura de computação em escala, usando uma linguagem de programação dedicada. Apesar de gerar benefícios para organizações de TI, os *scripts IaC* contêm segredos, como senhas codificadas. A prevalência de segredos em *scripts IaC* ressalta a importância de gerenciar adequadamente os segredos nesse tipo de scripts.

De acordo com essa pesquisa (Rahman *et al*., 2019), as práticas identificadas incluem o uso de ferramentas como, por exemplo, desbloqueio para Hashicorp Vault, bem como práticas independentes de ferramentas, como controle de acesso. As descobertas também revelam que, para gerenciamento secreto em *IaC*, os praticantes podem usar ferramentas específicas de linguagem − por exemplo, Hiera para Puppet −, bem como ferramentas como Hashicorp Vault, que funcionam com todas as linguagens de *IaC*.

Por fim, Rahman *et al*. (2019) relatam que as descobertas estabelecem as bases para pesquisas futuras relacionadas ao desenvolvimento seguro de scripts de *IaC*. O trabalho demonstra que existe espaço para estudos referentes a gerenciamento seguro de credenciais em ambientes de *IaC*.

Basak *et al*. (2023) fizeram um estudo comparativo de relatórios de segredos de software por ferramentas de detecção de segredos, e abordam uma avaliação de nove ferramentas de detecção de segredos, em uso atualmente no mercado. O objetivo da pesquisa é auxiliar os desenvolvedores na escolha da melhor ferramenta dentro do contexto de uso.

O estudo de Basak *et al.* (2023) permite evidenciar a necessidade de se ter mecanismos complementares para proteção de credenciais. Segundo os autores, utilizou-se um conjunto de dados de referência contendo 818 repositórios do GitHub. Descobriram que três principais ferramentas com base na precisão são: GitHub *Secret Scanner* (75%), Gitleaks (46%) e Commercial X (25%), e com base no recall são: Gitleaks (88%), SpectralOps (67%) e TruffleHog (52%).

Além disso, o estudo fornece um conjunto de dados de falsos positivos para agilizar a pesquisa em detecção de segredos. Também se caracterizaram os recursos oferecidos pelas ferramentas de detecção de segredos para ajudar a prevenir a exposição de segredos. Contudo, os autores recomendam que os desenvolvedores escolham ferramentas com base nos tipos de segredos presentes no projeto, para evitar segredos perdidos.

Na mesma linha, Basak *et al.* (2022), em artigo sobre práticas para gerenciamento de segredos em artefatos de *software*, tiveram como objetivo ajudar os profissionais de tecnologia da informação a evitar a exposição de segredos, identificando práticas no gerenciamento de credenciais em artefatos de *software*.

O estudo permitiu identificar vinte e quatro práticas agrupadas em seis categorias, indicadas por desenvolvedores e empresas. As descobertas indicam que usar variáveis de ambiente local e serviços externos de gerenciamento de segredos são as práticas mais recomendadas para mover segredos para fora do código-fonte e armazená-los com segurança. Também se observou que, ao usar ferramentas de varredura em sistemas de controles de versão, assim como empregar tempo de validade reduzido em segredos são as práticas mais recomendadas para evitar a exposição de credenciais.

Dentro da pesquisa, os autores (Basak *et al*., 2022) destacam quatro práticas recomendadas para limitar a exposição de segredos em ambientes *IaC*. São elas: a) utilizar segredos com curta duração, facilmente implementados e gerenciados por um cofre de senhas; b) restringir acesso a permissões de API. Dentro dessa prática, limitam-se o acesso e as permissões, restringindo o movimento lateral entre as aplicações; c) a terceira prática se refere a revogar segredos e higienizar controles de versão; d) a última se refere à auditoria e visibilidade de logs para atividades suspeitas.

Em outra pesquisa que discute a gestão de segredos, Prakash (2024) explora as limitações das práticas tradicionais de gerenciamento de segredos e articula os benefícios de uma abordagem unificada, fornecendo um modelo para proteger ambientes computacionais em nuvem, reduzindo complexidades operacionais e garantindo a conformidade.

O autor insere uma estratégia composta de práticas combinadas que incluem desde o uso de um cofre de senhas integrado com a plataforma em nuvem até a adoção de práticas da federação de carga de trabalho para acesso seguro, que engloba o uso do IAM nativo da nuvem, o uso do princípio menor privilégio e a duração curta das credenciais.

O trabalho de Prakash (2024) elucida como um dos principais riscos de segurança, o armazenamento de segredos em texto simples ou em locais não seguros, pode levar a acesso não autorizado e potenciais violações de segurança, aumentando o risco de exposição dos segredos.

De acordo com Oliveira *et al*. (2024), a implementação de aplicativos em um ambiente de execução confiável TEE (*Trusted Execution Environment*) apresenta possíveis complexidades para administradores de sistema, no que diz respeito à gestão de vulnerabilidades. A pesquisa demonstra que a utilização de um mecanismo de segurança baseado em OTP (*One-Time Password*) dentro dessa arquitetura não apenas elevou o nível de segurança do ambiente, mas também manteve a usabilidade e a produtividade dos usuários em níveis satisfatórios.

Entre os recursos explorados na pesquisa, destaca-se o uso da libOS Gramine SGX, que proporciona uma proteção em nível de *hardware*. Além disso, foi incorporado o cofre de senhas da Hashicorp Vault e um mecanismo de proteção desenvolvido em Python, utilizando as bibliotecas PyOTP e HVAC para facilitar a implementação da autenticação multifator (MFA) com integração do OTP.

Para validar os resultados da pesquisa, foram realizadas análises de segurança e testes de invasão, que indicaram uma melhoria significativa na segurança sem comprometer a validade temporal no PAM (*Privilege Access Management*), nem exigir alterações no código ou nas senhas.

Esta abordagem em camadas não apenas reforçou a segurança geral do sistema, mas também introduziu complexidades adicionais para administradores de sistema, especialmente em relação ao gerenciamento rotineiro de vulnerabilidades.

No seu trabalho sobre gerenciamento de segredo seguro, Blazej (2016) resume diferentes abordagens para a gestão de segredos compartilhada em sistemas, discutindo os principais recursos e a segurança. O autor analisa a questão de que o compartilhamento secreto precisa ser empregado na autenticação não-humana. Se um sistema, por exemplo, um servidor de aplicativos, precisa se autenticar em outro, por exemplo, um servidor de banco de dados, ele tem que fornecer as credenciais corretas, gerando um problema na distribuição dos segredos para os sistemas de acesso.

Nesse estudo, Blazej (2016) analisou detalhadamente os principais *softwares* para resolver questões de segredos compartilhados, verificando as principais lacunas relacionadas à segurança. Com isso, o autor apresentou uma lista de diretrizes e recomendações relacionadas à segurança, entre elas: o segredo nunca deve sair da máquina-cliente em texto simples; a comunicação com o repositório deve ser criptografada com um mecanismo comumente usado, como TLS, em vez de criar API de autenticação e sessão; todas as solicitações devem ser assinadas com o nome do usuário, chave privada e a API deve ser sem estado.

Por fim, Blazej (2016) ressalta a importância de se criar um modelo de sistema de compartilhamento de segredos, a relevância de um modelo de ameaças relacionado ao contexto, e apresenta uma abordagem de recomendação para um projeto de sistema de compartilhamento de segredos, dando base para futuras implementações e discussões.

Em um artigo recente da Codacy (2023), ressalta-se a importância do gerenciamento de segredos, fazendo um panorama do uso de segredos nas mais diversas aplicações, envolvendo desde ferramentas de automação, códigos em repositórios, soluções internas e comerciais, servidores, banco de dados em nuvem, aplicativos de *containers* e ferramentas de integração/implantação contínua CI/CD, entre outros.

Segundo a Codacy (2023), uma prática comum que dificulta o gerenciamento de segredos é a descentralização do processo pelas organizações, ao invés de centralizá-los em um único ambiente. Quando desenvolvedores, administradores, equipes de DevOps e

outros profissionais gerenciam segredos de forma independente, a visibilidade geral do ambiente é comprometida.

Essa abordagem fragmentada, baseada em silos, impede uma visão holística das camadas de infraestrutura. Como consequência, a auditoria de segredos torna-se mais complexa, aumentando as chances de ocorrência de vulnerabilidades e lacunas de segurança.

Concluindo, conforme destacado pela Codacy (2023), o crescente número de segredos utilizados em ferramentas e sistemas amplifica significativamente os desafios de gerenciamento de segredos, especialmente em ambientes DevOps. Nesse cenário, as equipes frequentemente dependem de dezenas de ferramentas e tecnologias para gerenciamento de configuração e orquestração, muitas vezes implementadas por meio de scripts automatizados.

Além disso, o artigo da Codacy (2023) destaca que o gerenciamento de segredos deve considerar contas de fornecedores terceirizados e soluções de acesso remoto, reforçando a necessidade de práticas robustas e centralizadas para mitigar riscos e garantir a segurança das operações.

Trabalhos recentes em segurança de *hardware* (Zhang, 2016), destacam três abordagens principais para proteção de circuitos integrados (CIs): ofuscação lógica combinacional, camuflagem de CIs e ofuscação baseada em componentes programáveis (PC). A ofuscação lógica combinacional insere portas de chave adicionais no *design* do CI, protegendo contra pirataria e construção excessiva, mas incorrendo em sobrecargas significativas de área e energia. Já a camuflagem de CIs utiliza técnicas de *layout*, como contatos fictícios e células padrão, para dificultar a extração da *netlist* de nível de porta.

Segundo Zhang (2016), embora seja eficaz em adicionar uma camada de defesa, essa técnica não protege adequadamente contra ataques em sistemas integrados. Por sua vez, a ofuscação baseada em PC substitui portas lógicas por componentes programáveis, como RAMs, configurados após a fabricação, mas também gera altas sobrecargas de desempenho devido às exigências adicionais de projeto.

Como solução, o trabalho desenvolvido por Zhang (2016) propõe uma técnica prática e eficiente de ofuscação lógica combinacional que supera as limitações existentes.

Essa abordagem impede que invasores infiram funções lógicas ao dificultar ataques de engenharia reversa baseados em processamento de imagem ou força bruta, enquanto mantém baixas sobrecargas de área e energia.

Além disso, o uso de respostas PUF (*Physical Unclonable Functions*) gera licenças dependentes de dispositivo, oferecendo proteção contra pirataria e construção excessiva. Experimentos mostram que a técnica proposta reduz significativamente as sobrecargas, com apenas 0,63% de aumento em área e 2,6% em energia nos benchmarks ISCAS, sem impacto no desempenho, oferecendo uma solução robusta e eficiente para proteção de IP em *hardware* (Zhang, 2016).

Por fim, o trabalho avalia estratégias de segurança baseadas em ofuscação, explorando possibilidades de proteção de circuitos integrados e *software* em *hardware*.

Neste capítulo, foram apresentados os principais trabalhos relacionados com o tema em estudo, que destacam contribuições relevantes para os o trabalho com *Zero Trust*, gestão de segredos e segurança em *IaC*.

Esses estudos reforçam a importância de se abordar os desafios de segurança em ambientes automatizados, evidenciando lacunas e oportunidades na integração de práticas seguras e no desenvolvimento de soluções voltadas para a proteção de credenciais, automação de infraestrutura e mitigação de vulnerabilidades.

O Quadro 2 resume as abordagens e os autores analisados, consolidando uma visão dos esforços existentes na área. A análise dos trabalhos reforça a necessidade de explorar novos modelos e técnicas que alinhem princípios de *Zero Trust* e automação segura, ampliando a proteção contra ameaças de cibersegurança e fortalecendo a confiabilidade das operações em ambientes computacionais modernos.

Quadro 2. Trabalhos relacionados e devidas características/abordagens

Características / Abordagens	Autores	Rahman, *et al.*, 2019	Rahman, *et al.*, 2021a	Konala, *et al.*, 2023	Ibrahim, *et al.*, 2022	Rahman e L. Williams, 2021	Rahman, *et al.*, 2021b	Basak, *et al.*, 2023	Basak, *et al.*, 2022	Prakash, 2024	Oliveira, *et al.*, 2024	Rong, *et al.*, 2022	Ahmadi, 2024	Blazej, 2016	Mehraj e Banday, 2020	Codacy, 2023	Zhang, 2015	**Modelo proposto**
Autenticação não-interativa											X							X
Aplicação e uso do *Zero Trust*											X	X	X					X
Aplicação e Análise de Segurança em *IaC*		X	X	X	X	X	X	X				X			X			X
Gestão de segredos / cofre de senhas						X	X		X	X	X			X		X		X
Aplicação e uso da Ofuscação																	X	X

Fonte: Do autor.

Capítulo 4

Modelo de arquitetura *Zero Trust* com *IaC* para autenticação não-interativa com ofuscação de credenciais

Este capítulo apresenta a arquitetura de segurança proposta, que integra tecnologias como infraestrutura como código, princípios de *Zero Trust*, cofre de senhas e técnicas de ofuscação de credenciais. A proposta visa criar um modelo de segurança em camadas, capaz de mitigar vulnerabilidades críticas e proteger credenciais sensíveis em ambientes automatizados, reduzindo a superfície de ataque e garantindo escalabilidade.

A estrutura do capítulo está organizada em três partes:

a) **Visão geral da proposta:** Apresenta a arquitetura e seus objetivos principais.
b) **Método proposto:** Descreve os componentes e técnicas utilizados no modelo.
c) **Validação do método:** Apresenta os resultados dos testes e análises, destacando eficácia e limitações.

A metodologia adotada combina experimentos práticos em ambiente controlado com análises qualitativas para validar a eficácia do modelo.

4.1 Visão geral da proposta

O modelo proposto tem como objetivo aprimorar a gestão e o armazenamento de segredos ou credenciais em cenários de autenticação não-interativa, onde não há intervenção humana no processo de autenticação. Esse tipo de autenticação é comum em integrações de sistemas, como comunicações entre APIs, serviços e bancos de dados, nos

quais as credenciais são armazenadas em arquivos de configuração, variáveis de ambiente ou embutidas no código-fonte das aplicações.

A ausência de interação humana torna a proteção dessas credenciais um desafio, pois qualquer comprometimento pode permitir que atacantes acessem recursos sensíveis sem necessitar de autenticação adicional. Para mitigar esse risco, o modelo proposto segue princípios de *Zero Trust* e implementa técnicas que reforçam a segurança em todas as etapas do ciclo de vida da credencial.

Este modelo se alinha às recomendações do NIST (*National Institute of Standards and Technology*), garantindo princípios como privilégios mínimos, distribuição de responsabilidades e monitoramento contínuo. Além disso, busca mitigar ameaças como movimento lateral, escalação de privilégios e exposição acidental de credenciais.

A proposta se divide em duas etapas:

a) **Modelo de Arquitetura *Zero Trust* baseada em *IaC*:**

Nesta etapa, é implementada uma arquitetura *Zero Trust* que aplica verificação contínua de identidade e postura de segurança. Toda interação entre sistemas é tratada como não confiável até que seja autenticada. A abordagem da *IaC* é utilizada para garantir reprodutibilidade, agilidade e segurança no provisionamento dos serviços. Além disso, são realizadas auditorias e verificações de vulnerabilidades em imagens de *containers* e serviços, assegurando conformidade com padrões de segurança e minimizando riscos de exposição.

b) **Gestão de Segredos com Ofuscação de Credenciais para Autenticação Não-Interativa:**

A segunda etapa introduz um mecanismo de gestão de segredos, incorporando técnicas de ofuscação de credenciais para evitar a exposição de informações sensíveis em texto claro. O modelo proposto utiliza OTP (*One-Time Password*) e HMAC (*Hash-Based Message Authentication Code*) para garantir que credenciais sejam protegidas contra ataques de repetição e força bruta. Além disso, a rotação periódica de chaves e a sincronização segura de contadores evitam comprometimentos por reutilização de credenciais.

Com essa abordagem, mitigam-se riscos como:

a) Exposição acidental de credenciais em código-fonte ou arquivos de configuração.
b) Ataques baseados na captura de credenciais armazenadas estaticamente.
c) Acesso não autorizado a serviços críticos por meio de credenciais comprometidas.

Ao implementar essa solução, a proposta mitiga as limitações da segurança baseada em perímetro, que frequentemente falha contra ameaças internas e movimentos laterais em redes corporativas.

4.2 Método proposto

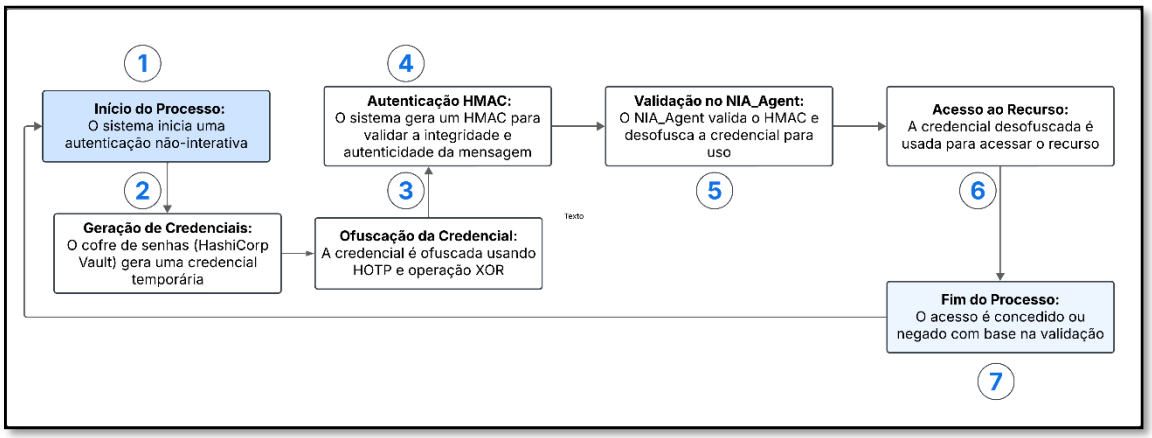
A implementação da arquitetura baseada em *Zero Trust* e *IaC* exige um conjunto de metodologias e práticas que garantem a integridade, autenticidade e proteção de credenciais em ambientes não-interativos. Neste contexto, o modelo proposto combina técnicas de segurança em camadas para mitigar vulnerabilidades associadas à exposição de credenciais e acessos indevidos.

A implementação do modelo envolve duas etapas:

a) **Arquitetura *Zero Trust* baseada em *IaC*:** Estabelece uma infraestrutura segura e auditável por meio de práticas automatizadas de provisionamento e gerenciamento, garantindo a aplicação rigorosa de políticas de segurança.
b) **Mecanismo de ofuscação e autenticação HMAC:** Implementa técnicas de proteção de credenciais, combinando ofuscação, autenticação baseada em OTP e validação com HMAC.

A Figura 3 apresenta um fluxograma que descreve etapas utilizadas no método proposto. O método utiliza tecnologias como o HashiCorp Vault para geração de credenciais temporárias, combinadas com técnicas de ofuscação e validação de integridade para garantir a segurança.

Figura 3 – Fluxograma com as etapas do método proposto



Fonte: Do autor.

A seguir, apresentam-se as etapas detalhadas:

1. **Início do Processo:** O sistema inicia uma autenticação não-interativa, onde uma aplicação solicita acesso a um recurso protegido.
2. **Geração de Credenciais:** O HashiCorp Vault gera uma credencial temporária, garantindo que ela seja única e segura.
3. **Ofuscação da Credencial:** A credencial gerada é ofuscada utilizando um algoritmo que combina HOTP e operação XOR, adicionando uma camada de segurança. Além da proteção da arquitetura e canal cifrado por TLS.
4. **Autenticação HMAC:** O sistema cria um HMAC para validar a integridade e a autenticidade da mensagem que contém a credencial.
5. **Validação no *NIA-Agent*:** O agente de autenticação (*NIA-Agent*) valida o HMAC e desofusca a credencial, tornando-a utilizável para o acesso.
6. **Acesso ao Recurso:** A credencial desofuscada é empregada para acessar o recurso protegido, como um banco de dados ou serviço de API.
7. **Fim do Processo:** O acesso é concedido ou negado com base na validação da credencial e na integridade dos dados.

4.2.1 Modelo de arquitetura *Zero Trust* baseada em *IaC*

A arquitetura *Zero Trust* redefine a segurança tradicional, assumindo que nenhuma conexão ou usuário deve ser confiável até que sua identidade e conformidade sejam verificadas.

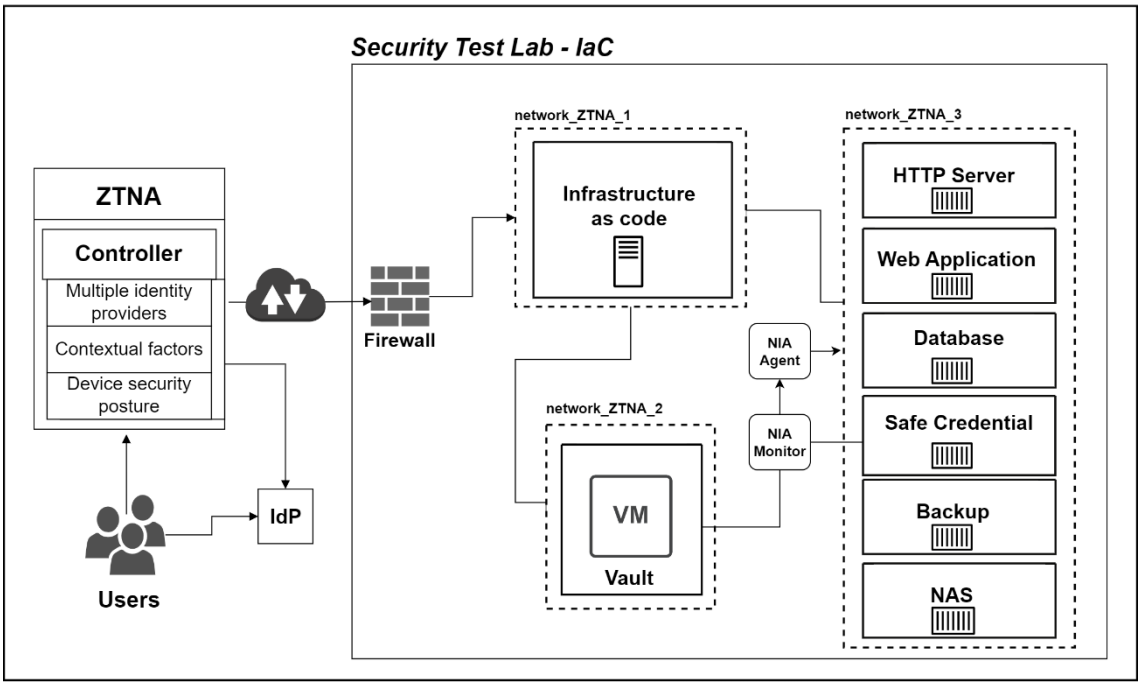
A integração da *IaC* nesse modelo permite que a infraestrutura de segurança seja declarativa e versionada, garantindo reprodutibilidade e conformidade contínua. No modelo proposto, a *IaC* é utilizada para:

a) Provisionar e configurar a infraestrutura de segurança.
b) Gerenciar acessos e permissões com princípios de privilégio mínimo.
c) Automatizar políticas de segurança e segmentação de rede.
d) Realizar auditoria contínua das configurações de segurança.
e) Integrar ferramentas de segurança como *scanners* de vulnerabilidade e controle de acesso baseado em identidade (IAM).

A Figura 4 apresenta uma visão geral da arquitetura, ilustrando os principais componentes e interações do modelo proposto. Esta configuração demonstra como o ZTNA, integrado à infraestrutura como código, fornece uma abordagem robusta para proteção de recursos e gestão de credenciais.

Essa arquitetura reflete os princípios do modelo *Zero Trust*, garantindo que cada interação seja verificada e que o acesso seja concedido apenas a entidades autenticadas e autorizadas.

Figura 4 – Visão geral da arquitetura proposta



Fonte: Do autor.

Para facilitar a compreensão da Figura 4, o Quadro 3 apresenta os principais elementos da arquitetura proposta, detalhando suas respectivas funções e descrevendo como cada componente contribui para a implementação dos princípios do modelo *Zero Trust*.

Quadro 3 – Principais elementos presentes na arquitetura proposta

Elemento	Descrição
ZTNA Controller	Componente responsável pelo gerenciamento centralizado de acesso baseado em políticas de *Zero Trust*. Ele verifica as identidades dos usuários através de múltiplos provedores de identidade, como SSO (*Single Sign-O*n) e OAuth, além de fatores contextuais, como localização, horário de acesso e tipo de dispositivo. Também monitora a postura de segurança do dispositivo (atualizações, antivírus e conformidade) antes de conceder acesso.
Firewall	Atua como uma barreira de proteção, aplicando políticas de controle de tráfego entre as diferentes redes internas e externas. No modelo *Zero Trust*, o *firewall* é configurado para liberar apenas portas e protocolos específicos, como TCP para SSH (porta 22) e HTTPS (porta 443). Bloqueia tráfego não autorizado em protocolos como UDP e ICMP para evitar tentativas de exploração e garantir uma superfície de ataque reduzida.
GLPI	Sistema de gestão de ativos e serviços de TI. O acesso ao GLPI é protegido pelo ZTNA *Controller*, que garante que apenas usuários e dispositivos autenticados possam interagir com o sistema. As políticas de segurança aplicadas restringem o tráfego às portas TCP autorizadas (por exemplo, 80 e 443), impedindo acesso não autorizado.
MariaDB	Banco de dados relacional utilizado para armazenar informações críticas, como dados de clientes e configurações de sistemas. O acesso ao MariaDB é restrito a serviços autenticados que passam pelo ZTNA *Controller* e pelo *firewall*. As conexões seguras são estabelecidas através de protocolos como SSL/TLS para garantir a proteção dos dados em trânsito.
Nginx	Servidor *web* que atua como um *proxy* reverso, gerenciando o tráfego de entrada e saída de aplicações web. O Nginx é configurado para aceitar apenas conexões seguras e autorizadas, utilizando certificados SSL/TLS e autenticação mútua, conforme as políticas de acesso definidas pelo ZTNA *Controller*.
NAS	Sistema de armazenamento em rede (*Network Attached Storage*) utilizado para armazenar arquivos e backups de dados. O acesso ao NAS é protegido por regras de acesso *Zero Trust*, garantindo que apenas dispositivos autenticados e em conformidade com as políticas de segurança possam acessar os dados.
SafeCredential	Mecanismo de segurança implementado para proteção de credenciais e segredos sensíveis, como chaves de API, tokens de acesso e senhas. É baseado em OTP, HMAC e utiliza ofuscação nas credenciais de acesso.

BACKUP	Serviço de backup que armazena cópias de segurança dos dados críticos da infraestrutura. O backup é configurado para ser realizado de forma automatizada e segura, com armazenamento protegido por criptografia e políticas de acesso *Zero Trust* para evitar acesso não autorizado.
Terraform	Ferramenta de *Infrastructure as Code (IaC)* utilizada para automatizar o provisionamento e a configuração de recursos na infraestrutura. No contexto *Zero Trust*, o Terraform é responsável por aplicar políticas de segurança diretamente no código, garantindo que as configurações de *firewall*, acesso e autenticação estejam em conformidade com os princípios de *Zero Trust* desde o provisionamento inicial.
Vault	Cofre de segredos utilizado para armazenar de forma segura chaves de API, credenciais e outros segredos sensíveis. O Vault permite a rotação automática de credenciais, reduzindo o risco de exposição de segredos. Ele também oferece controle de acesso granular baseado em políticas de *Zero Trust*, garantindo que apenas usuários e serviços autenticados possam acessar os segredos armazenados.

Fonte: Do autor.

Para maiores detalhes sobre a arquitetura *Zero Trust* baseada em *IaC*, recomenda-se consultar a Seção 2.2.1 do Capítulo 2 (Fundamentação), onde são abordados os principais conceitos da arquitetura proposta, as características mínimas do modelo *Zero Trust* consideradas e demais informações relevantes.

4.2.2 Aplicação de camadas de segurança em *IaC* e *Zero Trust*

A adoção de múltiplas camadas de segurança, combinando diversas tecnologias, apresenta uma abordagem adequada para proteger sistemas e dados contra ameaças de cibersegurança. Esta seção explora essas questões e apresenta boas práticas para mitigar os desafios associados ao uso do modelo de segurança em camadas, em arquiteturas baseadas em *IaC*.

A abordagem de segurança em camadas enfrenta ameaças complexas, protegendo redes, sistemas e dados de maneira abrangente. Tecnologias como *firewalls*, antivírus e criptografia compõem essas camadas, garantindo proteção em diversos níveis. Além disso, o modelo *Zero Trust*, que reforça a necessidade de validações contínuas em todas

as interações entre usuários, dispositivos e sistemas, é uma prática recomendada nesse contexto.

Embora a implementação de camadas de segurança combinadas e o uso de várias tecnologias possam introduzir complexidades e desafios, uma abordagem estratégica e gerenciada pode mitigar esses obstáculos, resultando em uma postura de segurança mais robusta e resiliente. Essa abordagem traz desafios adicionais quando implementada por meio de *IaC*:

a) **Complexidade operacional:** Gerenciar várias camadas de segurança no código aumenta a complexidade do ambiente. Cada camada exige validações específicas, o que pode dificultar o provisionamento de recursos e a manutenção contínua das políticas de segurança (NIST, 2020).
b) **Interoperabilidade:** Diferentes ferramentas e tecnologias utilizadas nas camadas de segurança podem não se integrar perfeitamente. Isso pode criar desafios para garantir que todas as camadas estejam funcionando de forma coesa em um ambiente automatizado (HashiCorp, 2024).
c) **Escalabilidade e padronização:** A generalização de políticas de segurança em um ambiente que utiliza *IaC* pode ser desafiadora, principalmente em arquiteturas multinuvem ou híbridas. Cada provedor de nuvem possui diferentes mecanismos de segurança, e garantir que as políticas sejam consistentes em todos os ambientes requer um gerenciamento cuidadoso (IBM, 2023b).

O modelo *Zero Trust* exige a implementação de múltiplas camadas de segurança para garantir a proteção dos recursos em todos os níveis, incluindo identidades, dispositivos, redes, aplicações e dados. Por meio da *IaC*, essas camadas podem ser automatizadas e gerenciadas de maneira eficiente, garantindo que as políticas de segurança sejam consistentes em diferentes ambientes.

Por exemplo, em um ambiente de *IaC*, é possível definir políticas de controle de acesso, criptografia de dados em repouso e em trânsito, autenticação multifatorial e segmentação de rede diretamente no código. Isso garante que, ao provisionar novos recursos, as políticas de segurança já estejam configuradas e aplicadas automaticamente.

A combinação de várias camadas de segurança dentro de um modelo *Zero Trust*, implementado por meio de *IaC*, é uma prática recomendada para aumentar a proteção contra ameaças de cibersegurança. Contudo, a abordagem apresenta desafios em termos de complexidade operacional, interoperabilidade e escalabilidade. Para mitigar esses desafios, é imprescindível adotar ferramentas de automação, monitoramento contínuo e políticas dinâmicas.

4.2.3 Redução da superfície de ataque na arquitetura proposta

A arquitetura proposta foi desenvolvida para minimizar a superfície de ataque, reduzindo os vetores exploráveis por adversários. Os pressupostos definidos no Capítulo 2 são aplicados da seguinte forma:

1. **Segurança de credenciais**
    - As credenciais são armazenadas no HashiCorp Vault, protegendo segredos contra exposição indevida.
    - O acesso aos segredos segue controle granular baseado em identidade, evitando permissões excessivas.
    - Os segredos são injetados dinamicamente, eliminando sua presença em código-fonte ou arquivos de configuração.

2. **Proteção contra engenharia reversa**
    - Credenciais são protegidas por ofuscação e HMAC, dificultando a extração de informações sensíveis.
    - O uso de OTPs dinâmicos impede reutilização de *tokens* interceptados por atacantes.

3. **Segmentação e controle de acesso**
    - A implementação de *Zero Trust Network Access* (ZTNA) impede acessos não autorizados, exigindo autenticação contínua.
    - A arquitetura utiliza microssegmentação de rede, garantindo que cada serviço tenha acesso apenas ao necessário.

Essas medidas garantem que qualquer tentativa de exploração seja limitada e facilmente detectada, reforçando a resiliência do sistema.

4.2.4 Gestão de segredos com ofuscação de credenciais para autenticação não-interativa

Esta seção apresenta o mecanismo de proteção de credenciais desenvolvido para autenticação não-interativa, permitindo maior segurança ao longo de todo o ciclo de vida da credencial.

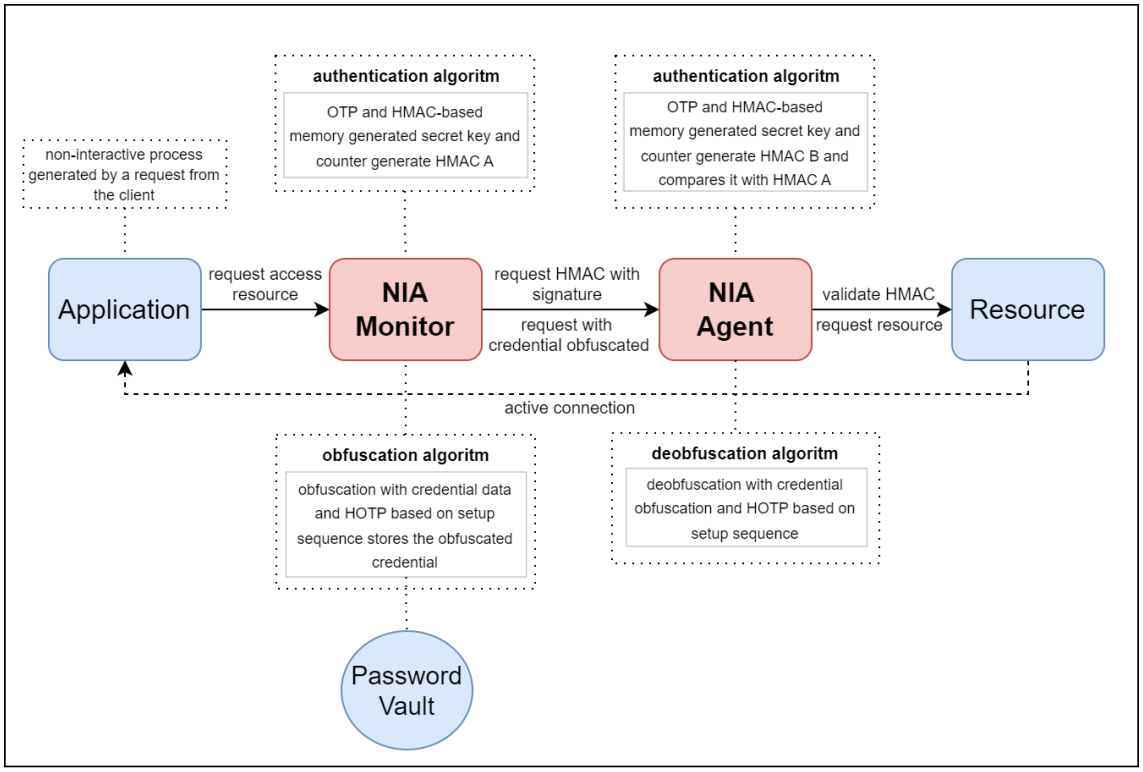
O mecanismo proposto utiliza técnicas de ofuscação de credenciais, que combinam dados da credencial com um código dinâmico (HOTP) para proteger informações sensíveis. O algoritmo de ofuscação integra-se ao cofre de senhas, garantindo que as credenciais sejam geradas e armazenadas de forma segura.

A Figura 5 apresenta uma visão geral do mecanismo de segurança (*SafeCredential*) e seus componentes. O fluxo básico ilustra as etapas de autenticação das mensagens entre as camadas de controle, representadas pelo *NIA-Monitor* e o *NIA-Agent*, bem como o processo de obtenção de credenciais no cofre de senhas e os procedimentos de ofuscação e desofuscação das credenciais.

Para facilitar o entendimento dos elementos que integram o mecanismo de segurança, as subseções seguintes fornecerão uma descrição detalhada dos dois principais artefatos do sistema:

a) **Algoritmo de ofuscação**
b) **Algoritmo de autenticação**

Figura 5 – Visão geral do mecanismo de segurança *SafeCredential*.



Fonte: Do autor.

Para melhor compreensão da Figura 5, o Quadro 4 apresenta uma descrição detalhada dos principais elementos que integram o mecanismo de segurança. Cada componente é descrito em termos de suas funções específicas e das interações entre si, com ênfase na gestão segura de credenciais e nos processos de autenticação realizados pelos artefatos que compõem a arquitetura proposta.

Quadro 4 – Elementos presentes no mecanismo de segurança

Elemento	Descrição
NIA-Monitor	Componente responsável por monitorar o fluxo de autenticação e ofuscação de credenciais. Ele faz requisições para o agente e realiza a autenticação inicial com HMAC (A). Recebe as requisições de uma API.
Authentication Algorithm (NIA-Monitor)	Algoritmo de autenticação implementado no monitor, que utiliza OTP e HMAC para gerar uma chave secreta baseada em memória e um contador. A saída é o HMAC A.
Obfuscation Algorithm	Algoritmo utilizado para ofuscar credenciais, combinando dados da credencial com HOTP baseado em uma sequência pré-definida. As credenciais são obtidas pela integração do algoritmo com o cofre de senhas (*password vault*). Realiza-se a ofuscação da credencial, não deixando o conteúdo em texto claro.

Password Vault	Sistema de gestão de segredos utilizados para geração de credenciais seguindo as boas práticas de segurança.
NIA-Agent	Componente responsável por processar solicitações de autenticação e desofuscação recebidas do monitor. Ele valida HMAC e trabalha com credenciais ofuscadas. O agente fica validando as requisições do monitor no lado do recurso.
Authentication Algorithm (NIA-Agent)	Algoritmo de autenticação no agente, que utiliza OTP e HMAC para gerar a chave secreta (HMAC B) e compará-la com o HMAC A recebido do monitor.
Deobfuscation Algorithm	Algoritmo que realiza a desofuscação das credenciais, utilizando HOTP e uma sequência configurada, permitindo acesso às credenciais originais.
Fluxo de Comunicação *(Monitor -> Agent)*	O monitor envia solicitações contendo HMAC e credenciais ofuscadas ao agente. O agente processa e valida os dados antes de realizar a desofuscação, em cada interação da requisição ao recurso.

Fonte: Do autor.

4.2.4.1 Algoritmo de ofuscação

O algoritmo de ofuscação se encarrega da integração e geração de credenciais no cofre de senhas e, ao retornar com a credencial de acesso, ao invés de trafegar ou armazenar os dados em texto simples/claro, essas credenciais são ofuscadas, dificultando o acesso ao seu conteúdo original.

Uma solução de gestão de segredos foi integrada ao algoritmo de ofuscação, permitindo uma proteção adicional às credenciais armazenadas. Essa integração fortalece a segurança ao combinar o armazenamento seguro de segredos com um mecanismo que dificulta a sua interpretação, mesmo em cenários de comprometimento parcial do sistema, garantindo maior resiliência contra ataques e acessos não autorizados.

O algoritmo utiliza a biblioteca de código aberto HVAC (*HashiCorp Vault API Client*), que possibilita a interação com o cofre de senhas HashiCorp Vault em Python, que podem ser consultados para maiores informação na seção 4.3.5.

A Figura 6 representa o fluxo de comunicação e as interações entre dois componentes principais do modelo proposto: o *NIA-Monitor* e o *NIA-Agent*. Esse

diagrama detalha como o mecanismo de ofuscação e autenticação segura funciona no contexto do gerenciamento de credenciais, detalhando os seguintes elementos e processos:

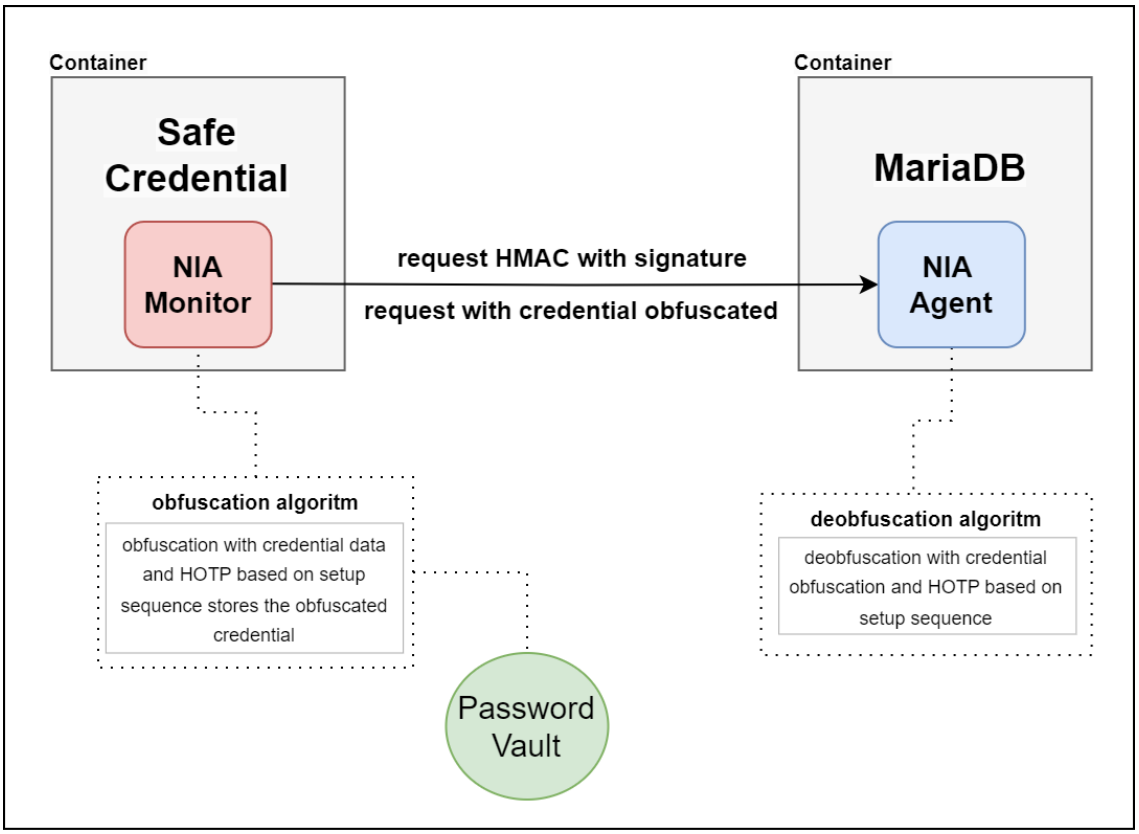
a) **O *NIA-Monitor*:**

1. Recupera a credencial do Password Vault.
2. Ofusca a credencial utilizando o algoritmo descrito.
3. Envia a requisição contendo o HMAC e a credencial ofuscada *ao NIA-Agent*.

b) **O *NIA-Agent*:**

1. Recebe a credencial ofuscada e realiza a desofuscação com o algoritmo correspondente.
2. Valida o HMAC e assegura que as credenciais estão corretas antes de autorizar o acesso.

Figura 6 – Fluxo do algoritmo de ofuscação



Fonte: Do autor.

O fluxo representado na imagem ilustra o processo de ofuscação e desofuscação de credenciais no contexto do modelo de segurança proposto. O *NIA-Monitor* atua como o componente central para obtenção de credenciais armazenadas no cofre de senhas, utilizando um algoritmo de ofuscação que combina os dados das credenciais com um

código dinâmico gerado por HOTP baseado em uma sequência de configuração pré-definida.

A credencial, após ser ofuscada, é enviada juntamente com um HMAC de autenticação para o *NIA-Agent*. Este, por sua vez, aplica o algoritmo de desofuscação e reverte o processo de ofuscação utilizando a mesma lógica e sequência. Ao final, o *NIA-Agent* faz a validação da autenticidade da credencial ofuscada e o HMAC recebido, assegurando que somente credenciais legítimas e protegidas sejam utilizadas.

Este fluxo reforça a segurança ao evitar a exposição de credenciais em texto claro e alinhar-se aos princípios do *Zero Trust*, promovendo a integridade e a proteção dos dados em trânsito e repouso.

A seguir, serão detalhados os procedimentos relacionados aos elementos do algoritmo de ofuscação, com o objetivo de oferecer uma compreensão clara e detalhada do seu funcionamento e das suas interações dentro do mecanismo de segurança.

O processo de ofuscação de credenciais converte um texto claro em uma sequência de caracteres codificada, tornando-o ilegível. A Figura 6 apresenta o pseudocódigo do processo, no qual o valor do HOTP, gerado com base em uma sequência ou contador armazenado em memória, é utilizado como chave para a codificação.

Figura 7 – Pseudocódigo do processo para ofuscar credencial

Algorithm 1 Obfuscate Credential Function

Require: credential: The original credential value (string)
Require: hotp: The HOTP value (string or numeric value)
Ensure: obfuscated_credential: The obfuscated credential value (Base64 encoded string)

1: Initialize an empty array to store the obfuscated bytes.
2: **for** each index i in the credential bytes **do**
3: Retrieve the current byte from the *credential* and the corresponding byte from the *HOTP* (repeat the HOTP if necessary).
4: Apply the XOR operation between the current byte and the HOTP byte.
5: Append the result to the array of obfuscated bytes.
6: **end for**
7: Encode the array of obfuscated bytes in Base64 format for easy storage.
return obfuscated_credential

Fonte: Do autor.

No processo de ofuscação, a função utiliza a operação XOR para combinar cada caractere da credencial (a ser protegida) com o valor correspondente do HOTP (um código dinâmico ou temporário usado como chave de ofuscação). O HOTP é gerado com

base em um contador iniciado manualmente em um processo interativo e armazenado em memória.

Cada caractere da credencial é convertido em seu valor ASCII (um número binário que representa o caractere). Esse valor é então combinado, por meio da operação XOR, com o valor ASCII do caractere correspondente no HOTP. Caso o HOTP seja mais curto que a credencial, ele é repetido ciclicamente para garantir que todos os caracteres sejam devidamente ofuscados, proporcionando segurança uniforme ao longo de toda a credencial.

O próximo passo consiste em converter os valores em *bytes* para um formato seguro e adequado para armazenamento ou transmissão. Nesse contexto, utiliza-se a codificação Base64, uma técnica que transforma dados binários em caracteres ASCII. Essa codificação converte os dados resultantes da operação XOR em uma *string* composta apenas por letras, números e alguns símbolos específicos, facilitando a sua manipulação e armazenamento.

Embora a codificação Base64 não seja uma forma de criptografia (não garante confidencialidade), ela desempenha um papel necessário ao transformar a sequência binária gerada pelo processo de ofuscação em um formato legível e reversível, permitindo a recuperação do conteúdo original quando necessário.

A Figura 8 apresenta o pseudocódigo para o processo de desofuscação da credencial, que segue a mesma lógica empregada na etapa de ofuscação, garantindo a consistência e integridade das informações.

Figura 8 – Pseudocódigo do processo para desofuscar credencial

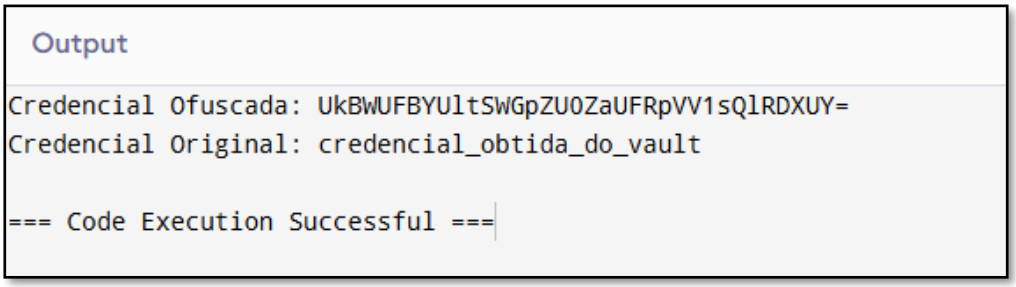
Algorithm 2 Deobfuscate Credential Function

Require: obfuscated_credential: The obfuscated credential value (Base64 encoded string)
Require: hotp: The HOTP value (string or numeric value)
Ensure: original_credential: The original credential value (string)
1: Decode the *obfuscated_credential* from Base64 to retrieve the obfuscated bytes.
2: Initialize an empty array to store the original bytes.
3: **for** each index i in the obfuscated_credential bytes **do**
4: Retrieve the current byte from the *obfuscated_credential* and the corresponding byte from the *HOTP* (repeat the HOTP if necessary).
5: Apply the XOR operation between the current byte and the HOTP byte.
6: Append the result to the array of original bytes.
7: **end for**
8: Convert the array of original bytes back into a readable string. **return** original_credential

Fonte: Do autor.

A Figura 8 ilustra o resultado do processo de ofuscação, apresentando a credencial transformada em uma *string* não legível, obtida após a aplicação do algoritmo de ofuscação. Além disso, destaca o valor original da credencial em texto simples, permitindo a visualização do contraste entre a forma protegida e a forma legível dos dados.

Figura 9 – Credencial ofuscada e com o seu conteúdo original



Fonte: Do autor.

4.2.4.2 Algoritmo de autenticação

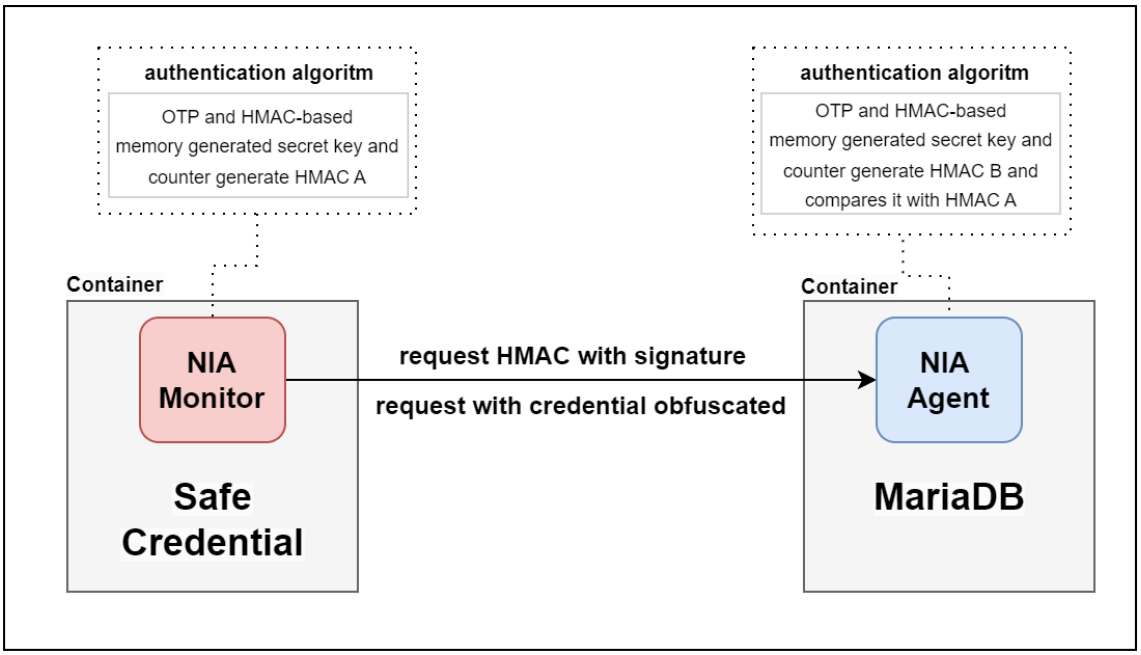
O segundo componente do mecanismo de segurança é o algoritmo de autenticação, projetado para validar a identidade de uma entidade, como usuário e sistema ou para assegurar a sua legitimidade. Esse algoritmo emprega métodos de verificação baseados em OTP e HMAC, garantindo tanto a autenticidade quanto a integridade das mensagens trocadas durante o processo de autenticação.

No esquema proposto, a autenticação é realizada entre camadas não-interativas para cada solicitação de recurso. Na Figura 10, essas camadas são representadas pelas entidades *NIA-Monitor* e *NIA-Agent*. A camada de controle *NIA-Agent* é acionada por um processo interativo, conduzido por um operador ou administrador do sistema, garantindo que os elementos críticos sejam configurados com segurança antes do início do fluxo não-interativo.

Esse procedimento é executado paralelamente ao processo de ofuscação, permitindo que a mensagem seja validada antes de conceder acesso ao recurso solicitado, e posteriormente em toda interação entre a aplicação e o banco de dados.

A Figura 10 representa o mecanismo seguro de autenticação que utiliza uma combinação de OTP e HMAC, com a geração e comparação de valores HMAC entre dois componentes do sistema. Este fluxo ajuda a garantir a integridade e autenticidade dos dados e a confirmação de que as credenciais e chaves usadas são legítimas.

Figura 10 – Fluxo do algoritmo de autenticação



Fonte: Do autor.

Este fluxo demonstra como o sistema utiliza um mecanismo de autenticação baseado em HMAC para proteger as credenciais ofuscadas contra alterações ou interceptações maliciosas. A sincronização dos contadores e a geração dinâmica dos HMACs são elementos-chave para garantir a segurança e a confiabilidade do processo.

a) **O *NIA-Monitor***:

1. Gera o HMAC A com base em sua lógica interna.
2. Envia para o *NIA-Agent*:
3. O HMAC A.
4. A credencial ofuscada.

b) **O *NIA-Agent*:**

1. Recebe os dados enviados.
2. Gera o HMAC B localmente utilizando a mesma sequência e contador.
3. Compara o HMAC B com o HMAC A.
4. Se forem iguais, a credencial é considerada válida.
5. Caso contrário, a requisição é rejeitada.

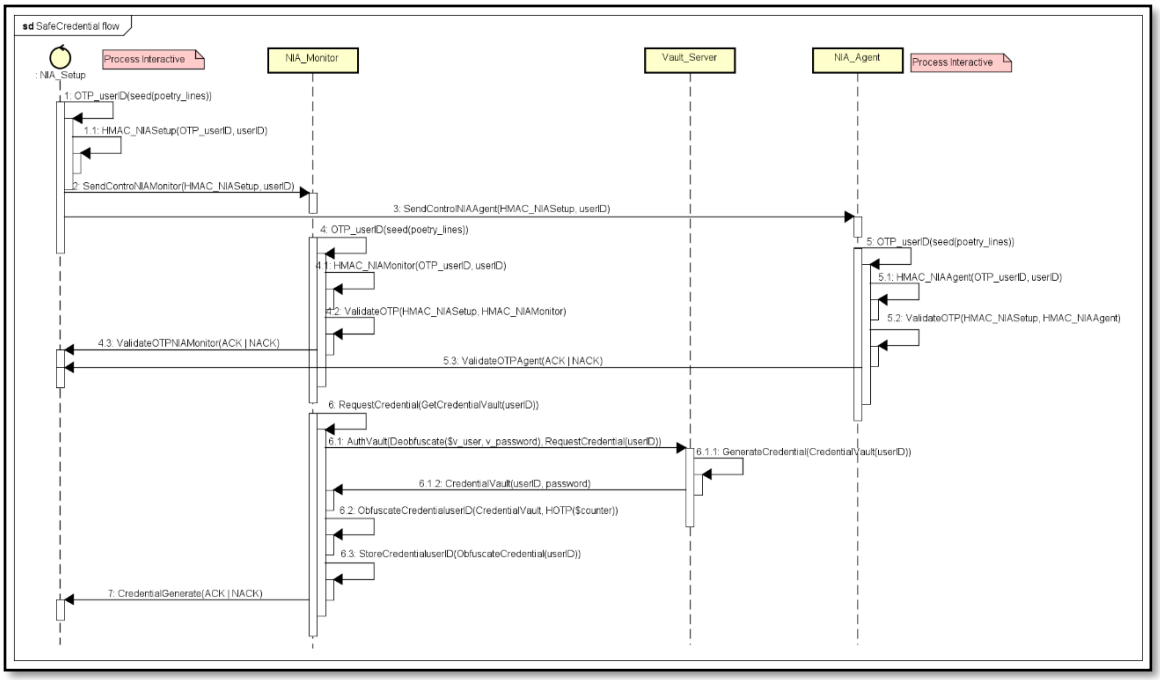
A Figura 11 apresenta um diagrama de sequência que ilustra o fluxo principal do mecanismo de segurança e as interações entre os componentes. O diagrama destaca os processos interativos necessários para a inicializar o mecanismo, incluindo a geração de sequências OTP, chaves secretas ou *seeds*, e outros procedimentos necessários.

Conforme o fluxo descrito na Figura 10, o processo inicia com a geração do OTP para o usuário de serviço solicitado, identificado como *userID* (sequência 1). Em seguida, realiza-se a geração do HMAC (sequência 1.1) e o envio do HMAC junto com o identificador do usuário (sequência 2). Na camada de controle *NIA-Monitor*, ocorre a validação do HMAC recebido (sequência 4.2), comparando-o com o HMAC gerado localmente. Esse processo de validação é replicado em todas as camadas de controle, garantindo a autenticação e a integridade do fluxo das mensagens.

No diagrama de sequência, destaca-se que o OTP_userID não é transmitido entre as camadas de controle; ele é gerado em tempo de execução a partir do userID e da semente (seed) local em cada camada. Essa estratégia aumenta significativamente a segurança, pois elimina a necessidade de transmissão direta do OTP entre as camadas.

A camada de controle *NIA-Setup* é responsável por inicializar o contador (usado na geração do HOTP) e configurar a semente ou chave secreta. Esse processo segue a lógica definida na função *poetry_lines*, que utiliza linhas de um texto (como um poema) somadas ao número do OTP gerado em tempo de execução. Essa abordagem assegura uma semente dinâmica e consistente em todas as camadas de controle *NIA*, reforçando a segurança e a integridade do sistema.

Figura 11 – Diagrama de sequência do mecanismo de segurança proposto



Fonte: Do autor.

A Figura 12 apresenta um pseudocódigo que detalha o processo de geração da semente (*seed*) em tempo de execução, demonstrando como a combinação de variáveis dinâmicas garante a consistência e robustez do mecanismo de autenticação.

Figura 12 – Processo de geração da *seed* em tempo de execução

Algorithm 3 Generate Seed Function

Require: poem_file: A text file containing a poem by Carlos Drummond de Andrade

Ensure: seed: A secret key to be used as the seed for the obfuscation algorithm

1: Call the *poetrylines* function to read the poem from the text file and load it into memory.
2: Initialize the seed construction.
3: Use the first 8 characters of the first line.
4: Use the last character of the first line.
5: Use the middle character of the second line.
6: Use the first two characters of the last line.
7: Combine these components into a single string to be used as the seed. **return** seed

Fonte: Do autor.

Para a geração do OTP em todo o processo, utiliza-se como chave secreta ou *seed* (semente) a composição de vários trechos de uma poesia do Carlos Drummond de Andrade (conforme se detalha na figura 8), presente em um arquivo TXT, o que pode dificultar o entendimento por parte de um agente malicioso.

No mecanismo de segurança, todos os procedimentos são executados em tempo real, sem armazenar chaves secretas ou valores gerados durante o processo. Dessa forma, torna-se significativamente mais complexo para um agente malicioso obter o valor original da credencial, pois além de superar as camadas impostas pela arquitetura *Zero Trust*, o invasor precisa:

a) Descobrir a lógica usada para compor a semente (*seed*).
b) Identificar a sequência exata do contador em execução para recriar o HOTP gerado.
c) Obter o valor do HMAC, o que exigiria depurar o código para encontrar a chave secreta, composta pela credencial e o valor do HOTP, além de determinar qual algoritmo de *hash* foi utilizado.
d) Realizar um *dump* de memória para acessar os dados, sendo que, a cada acesso, o OTP é alterado, gerando novos valores de autenticação. Para acessar a memória, o invasor precisaria de privilégios elevados e a capacidade de se mover lateralmente entre as entidades.

Essa abordagem aumenta a segurança, dificultando o acesso aos dados sensíveis em caso de tentativa de invasão. O processo de ofuscação se diferencia da criptografia tradicional, uma vez que não utiliza chaves públicas ou privadas, evitando o desafio relacionado ao armazenamento seguro dessas chaves para os processos de criptografar e descriptografar. Trata-se de uma alternativa viável no contexto proposto, pois está protegido pelas camadas impostas pela arquitetura *Zero Trust*.

O processo de ofuscação em algumas linguagens de programação é um procedimento padrão, voltado para a segurança (oculta dados sensíveis, proteção de propriedade e dificulta a engenharia reversa) e otimização, no intuito de diminuir o tamanho do código-fonte.

Além dos princípios de *Zero Trust*, a geração dinâmica de credenciais foi incorporada à arquitetura como uma prática que reforça a segurança, garantindo que as credenciais sejam criadas sob demanda e possuam validade limitada. Essa abordagem reduz significativamente os riscos de exposição e uso indevido, fortalecendo a proteção em ambientes dinâmicos e automatizados.

Ao adotar boas práticas e uma abordagem unificada, a arquitetura proposta eleva significativamente os padrões de segurança, garantindo conformidade e resiliência frente às ameaças de cibersegurança.

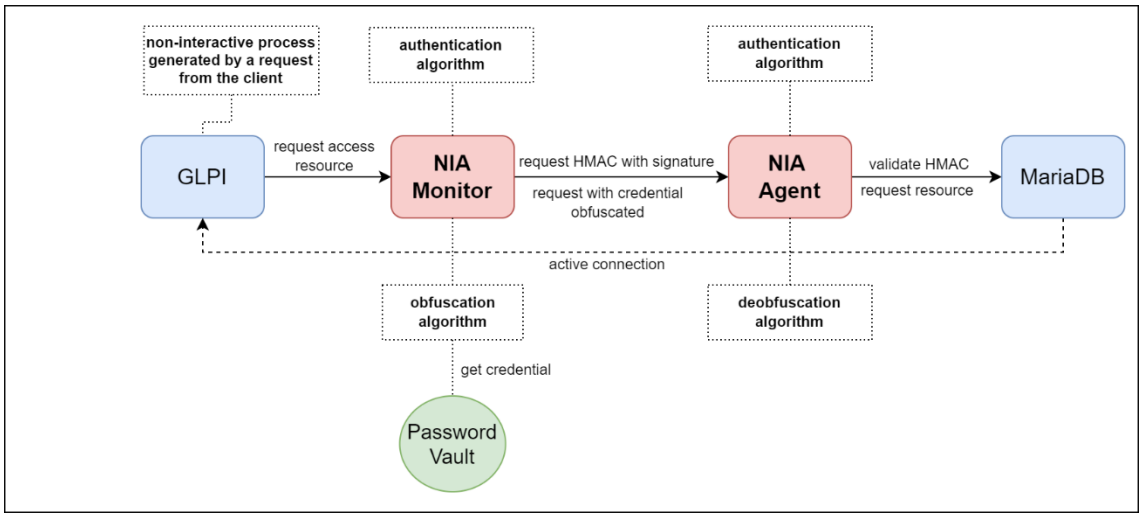
4.2.4.3 Interação dos componentes do mecanismo de proteção

A interação entre os componentes do mecanismo de proteção adiciona uma camada de segurança ao processo de autenticação em cenários não-interativos, por meio da validação HMAC implementada pelo algoritmo de autenticação.

A Figura 13 apresenta o fluxo principal do mecanismo de segurança, destacando as interações entre seus componentes, complementadas pelas Figuras 14, 15 e 16. Essas representações oferecem uma visão comparativa entre um processo tradicional de autenticação, que se baseia em um cofre de senhas para gerenciar credenciais e acessar recursos, e o procedimento aprimorado pelo mecanismo de proteção proposto.

A proposta aumenta o nível de segurança e reduz os riscos associados ao comprometimento de credenciais em cenários de autenticação automatizada.

Figura 13 – Fluxo principal do mecanismo de segurança proposto com interações



Fonte: Do autor.

Como exemplo prático, considera-se a aplicação GLPI, que requer acesso a um recurso, neste caso, um banco de dados MariaDB. Nesse fluxo, a credencial gerada pelo cofre de senhas possui um tempo de validade limitado a 8 horas.

Embora essa abordagem já represente um avanço em relação a métodos estáticos de autenticação, ela ainda expõe vulnerabilidades. Caso um *insider* comprometa essa credencial, um atacante poderia explorá-la para obter acesso prolongado ao recurso, aumentando os riscos de exploração e comprometimento de dados sensíveis.

O mecanismo proposto mitiga esses riscos ao integrar métodos como geração dinâmica de credenciais e autenticação baseada em OTP e HMAC, garantindo que as credenciais sejam utilizadas apenas de forma temporária e sob condições estritamente controladas. Isso reduz o impacto de potenciais comprometimentos e eleva a resiliência da arquitetura frente a ameaças internas e externas.

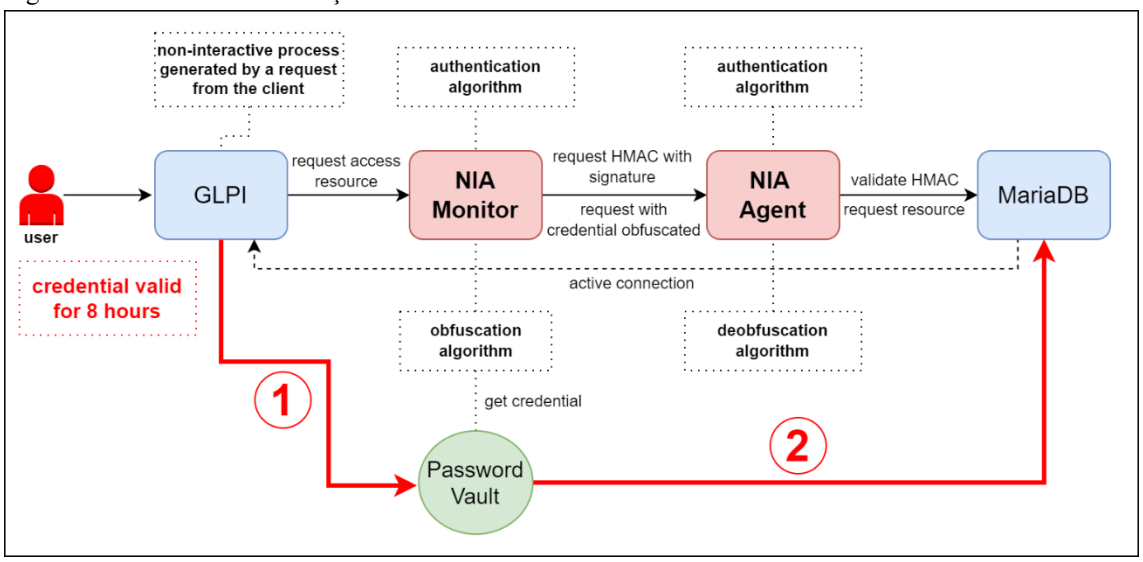
Conforme discutido anteriormente, a Figura 14 apresenta o exemplo de um fluxo em um modelo tradicional de autenticação não-interativa, destacando a interação entre os componentes GLPI, o cofre de senhas e o banco de dados MariaDB.

O processo segue duas etapas:

a) **Etapa 1:** O GLPI solicita e obtém uma credencial gerada pelo cofre de senhas (Vault), com validade de 8 horas. Essa credencial pode ser configurada diretamente em um arquivo local ou disponibilizada via API.
b) **Etapa 2:** A credencial é utilizada para autenticar a aplicação no banco de dados MariaDB, permitindo acesso ao recurso enquanto estiver dentro do prazo de validade.

O modelo tradicional, portanto, destaca a importância de evoluir para soluções que minimizem a exposição de credenciais, reforçando os mecanismos de segurança e garantindo maior proteção contra possíveis comprometimentos.

Figura 14 – Fluxo de autenticação não-interativa com o modelo tradicional



Fonte: Do autor.

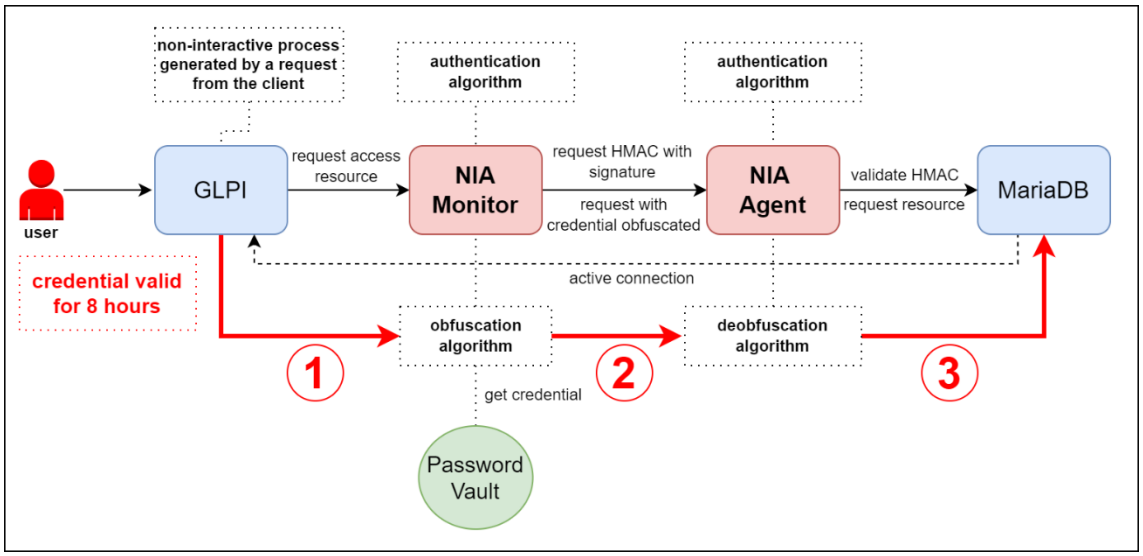
A Figura 15 apresenta o fluxo de autenticação não-interativa com o mecanismo de segurança proposto, destacando o processo de autenticação e a interação entre os componentes envolvidos.

O modelo é estruturado em três etapas principais, detalhando os procedimentos de acesso do GLPI, que ocorrem exclusivamente por meio do *NIA-Monitor* e do *NIA-Agent*. Nesse contexto, o GLPI não possui acesso direto ao banco de dados ou ao cofre de senhas, uma vez que ambos são gerenciados integralmente pelo mecanismo de segurança.

Após a credencial ser gerada pelo cofre de senhas, ela passa por um processo de ofuscação, garantindo que não permaneça em texto claro durante seu ciclo de validade, que é limitado a 8 horas. Isso reduz significativamente os riscos de exposição em caso de acesso não autorizado.

Além disso, a autenticação HMAC é aplicada a cada interação entre a aplicação e o banco de dados. Isso implica que, mesmo que a credencial ofuscada seja comprometida, o acesso ao banco de dados dependerá de uma validação adicional por meio do HMAC, adicionando uma camada extra de segurança.

Figura 15 – Fluxo de autenticação não-interativa com o modelo proposto



Fonte: Do autor.

Esse modelo proporciona maior robustez ao processo de autenticação, fortalecendo proteção nas interações entre a aplicação e os recursos protegidos, ao mesmo tempo em que reduz a superfície de ataque associada à gestão de credenciais.

No capítulo seguinte, será apresentada a implementação prática em um protótipo funcional, com o objetivo de demonstrar os recursos e procedimentos descritos até aqui. Essa etapa busca exemplificar detalhadamente o funcionamento da proposta, evidenciando a sua aplicação o e benefícios em cenários reais.

4.2.4.4 Integração com o cofre de senhas (Vault)

O uso de um cofre de senhas é fortemente recomendado pelo CIS Controls v8, uma estrutura amplamente reconhecida que fornece um conjunto priorizado de melhores práticas em matéria de cibersegurança; está projetado para lidar com ameaças contemporâneas a sistemas e *softwares* (CIS Controls, c2025).

De acordo com o Control 06 - *Access Control Management* do CIS Controls, é indicado o uso de processos e ferramentas que permitam criar, atribuir, gerenciar e

revogar credenciais de acesso e privilégios para contas de usuários, administradores e serviços, cobrindo ativos e *softwares*.

Nesse contexto, o cofre de senhas é indispensável para o mecanismo de segurança, oferecendo uma abordagem centralizada para a gestão de segredos. Seja em ambientes *on-premises*, em nuvem, ou híbridos, ele viabiliza uma gestão robusta e segura das credenciais. Na solução proposta, o acesso ao HashiCorp Vault é realizado pelo mecanismo de segurança, que gerencia a interação com as credenciais, garantindo a autenticidade, confiabilidade e integridade das informações.

A integração do cofre de senhas com o mecanismo de segurança não apenas centraliza o armazenamento de segredos, mas também opera como um PAM (*Privileged Access Management*). Essa combinação permite uma gestão eficiente da geração de credenciais e do controle de acesso, além de facilitar auditorias e a aplicação de políticas de segurança rigorosas.

Essa ferramenta oferece uma integração simplificada para desenvolvedores e administradores de sistemas, permitindo incorporar o gerenciamento seguro de segredos e a proteção de dados em aplicativos e processos automatizados de forma eficiente e confiável.

No capítulo 5 – protótipo, descreve-se o processo de integração entre o mecanismo de segurança e o cofre de senhas.

4.2.4.5 Análise de segurança

Nesta seção realiza-se uma análise de segurança, com o objetivo de garantir que o sistema atenda às propriedades de segurança desejadas, como confidencialidade, integridade, autenticidade e resistência a ataques de repetição.

a) **Definição do fluxo de segurança**

O fluxo de segurança envolve os seguintes componentes e etapas:

1. *NIA-Monitor*:

- Gera uma credencial temporária.

- Ofusca a credencial usando HOTP (*HMAC-Based One-Time Password*) e uma operação XOR para combinar a credencial com o valor do HOTP.
- Gera um HMAC (*Hash-based Message Authentication Code*) para validar a integridade e autenticidade da mensagem. O HMAC é calculado usando uma chave secreta compartilhada entre o *NIA-Monitor* e o *NIA-Agent*.
- Envia a credencial ofuscada e o HMAC para o *NIA-Agent*.

2. *NIA-Agent*:
   - Recebe a credencial ofuscada e o HMAC.
   - Valida o HMAC para garantir que a mensagem não foi alterada durante a transmissão e que foi enviada pelo *NIA-Monitor*.
   - Desofusca a credencial usando o mesmo valor de HOTP e a operação XOR.
   - Usa a credencial desofuscada para acessar o recurso protegido.

b) **Propriedades de segurança verificadas**

O fluxo de segurança envolve os seguintes componentes e etapas:

1. Confidencialidade:
   - A credencial ofuscada não revela informações sobre a credencial original. A ofuscação é realizada usando uma operação XOR com um valor dinâmico (HOTP), garantindo que a credencial não possa ser recuperada sem o valor correto do HOTP.
   - A credencial só pode ser desofuscada pelo *NIA-Agent*, que possui a chave correta (HOTP).
2. Integridade:
   - O HMAC garante que a mensagem não foi alterada durante a transmissão. O *NIA-Agent* valida o HMAC antes de desofuscar a credencial, garantindo que a mensagem seja íntegra.
3. Autenticidade:
   - Apenas o *NIA-Monitor* e o *NIA-Agent* podem gerar e validar o HMAC. Isso garante que a mensagem foi enviada pelo *NIA-Monitor* e não por um atacante.
4. Resistência a ataques de repetição:

- O uso de OTP garante que cada mensagem seja única e válida apenas por um curto período de tempo. Isso impede que um atacante reutilize uma mensagem autenticada para obter acesso não autorizado.

Nenhuma vulnerabilidade crítica foi identificada durante a análise de segurança, demonstrando que o fluxo de segurança é robusto e atende aos requisitos de segurança.

4.2.4.6 Superfície de ataque

Nesta seção, a superfície de ataque do sistema é definida e expandida, identificando os pontos onde um atacante poderia tentar explorar vulnerabilidades. A superfície de ataque inclui:

a) **Componentes técnicos**

1. APIs e serviços:
   - As APIs que permitem a comunicação entre o *NIA-Monitor*, *o NIA-Agent* e o HashiCorp Vault são pontos críticos de exposição. Um atacante poderia tentar interceptar ou manipular as requisições para obter acesso não autorizado.
   - Exemplo de ataque: Um atacante poderia realizar um ataque de *Man-in-the-Middle* (MITM) para interceptar as credenciais ofuscadas e o HMAC.
2. Banco de dados:
   - O banco de dados MariaDB que armazena informações sensíveis, é um alvo potencial para ataques de injeção SQL ou acesso não autorizado.
   - Exemplo de ataque: Um atacante poderia explorar uma vulnerabilidade de injeção SQL para acessar dados sensíveis.
3. Cofre de segredos (HashiCorp Vault):
   - Embora o Vault seja projetado para ser seguro, configurações incorretas ou políticas de acesso inadequadas podem expor credenciais sensíveis.
   - Exemplo de ataque: Um atacante com acesso ao Vault poderia tentar extrair credenciais armazenadas.

4. *Containers* e imagens:

- Os containers que hospedam os serviços podem conter vulnerabilidades conhecidas ou configurações inseguras.
- Exemplo de ataque: Um atacante poderia explorar uma vulnerabilidade em uma imagem de contêiner para obter acesso ao sistema.

b) **Processos de autenticação e gestão de segredos**

1. Autenticação não-interativa:

- O processo de autenticação não-interativa, onde as credenciais são geradas e ofuscadas automaticamente, pode ser alvo de ataques de repetição ou interceptação.
- Exemplo de ataque: Um atacante poderia capturar uma mensagem autenticada e tentar reutilizá-la.

2. Ofuscação de credenciais:

- Um atacante poderia tentar reverter o processo de ofuscação se o algoritmo utilizado for fraco ou se a chave de ofuscação (HOTP) for comprometida.
- Exemplo de ataque: Um atacante poderia realizar um ataque de força bruta para descobrir a credencial original.

3. Rotação de chaves:

- Se o processo de rotação não for seguro, ele pode se tornar um vetor de ataque.
- Exemplo de ataque: Um atacante poderia explorar uma falha no processo de rotação para obter acesso a credenciais antigas.

c) **Infraestrutura de rede e comunicação**

1. Comunicação entre componentes:

- A comunicação entre o *NIA-Monitor*, o *NIA-Agent* e outros serviços ocorre através de redes que podem ser alvo de ataques de *sniffing* ou *Man-in-the-Middle* (MITM).
- Exemplo de ataque: Um atacante poderia interceptar o tráfego de rede para capturar credenciais ofuscadas.

2. *Firewall* e ZTNA:

- Configurações inadequadas podem deixar portas abertas para ataques.

- Exemplo de ataque: Um atacante poderia explorar uma porta aberta no *firewall* para obter acesso não autorizado.

3. *Logs* e monitoramento:

- Os *logs* gerados pelo sistema podem conter informações sensíveis.
- Exemplo de ataque: Um atacante poderia acessar os *logs* para obter informações sobre o sistema.

d) Estratégias de mitigação

Para reduzir a superfície de ataque, são implementadas várias camadas de segurança, incluindo:

1. Segmentação de rede e controle de acesso:

- O uso do ZTNA Controller e do *firewall* permite a segmentação granular da rede, garantindo que apenas os serviços autorizados possam se comunicar entre si.
- Exemplo: As políticas de acesso são definidas com base no princípio de privilégio mínimo, onde cada componente tem acesso apenas aos recursos necessários para sua operação.

2. Proteção de credenciais e segredos:

- As credenciais são armazenadas no HashiCorp Vault, que oferece armazenamento seguro e rotação automática de chaves.
- Exemplo: O Vault é configurado para rotacionar as credenciais periodicamente, reduzindo o risco de comprometimento.

3. Autenticação forte e dinâmica:

- O uso de OTP e HMAC garante que as credenciais sejam válidas apenas por um curto período de tempo.
- Exemplo: Cada mensagem autenticada é única e válida apenas por um curto período de tempo, impedindo ataques de repetição.

4. Monitoramento contínuo e resposta a incidentes:

- O sistema é monitorado continuamente para detectar atividades suspeitas.
- Exemplo: Em caso de detecção de um comprometimento, o sistema inicia automaticamente o processo de rotação de chaves e recuperação de segurança.

4.2.4.7 Mapeamento de ameaças e mitigações na arquitetura proposta

Nesta seção, apresenta-se o mapeamento de ameaças da arquitetura proposta, utilizando a metodologia STRIDE para identificar possíveis vulnerabilidades e riscos de segurança. O Quadro 5 resume as principais ameaças identificadas, seus impactos, probabilidades e as mitigações implementadas na arquitetura.

Quadro 5 – Modelo de ameaças na arquitetura proposta

Componente	Ameaça	Impacto	Probabilidade	Mitigação
ZTNA Controller	*Spoofing* (falsificação de identidade)	Alto	**Média**	Autenticação multifator (MFA) e validação contínua de identidade.
NIA Monitor/Agent	Interceptação de credenciais	Alto	**Alta**	Ofuscação de credenciais e uso de HMAC para integridade das mensagens.
HashiCorp Vault	Acesso não autorizado	Crítico	**Baixa**	Políticas de acesso granular e rotação automática de credenciais.
Banco de Dados MariaDB	Injeção de SQL	Alto	**Média**	Uso de *prepared statements* e validação de entradas.

Fonte: Do Autor.

a) Análise das Ameaças:

- ***Spoofing*:** A falsificação de identidade pode permitir que um atacante se passe por um usuário legítimo e acesse recursos protegidos. Para mitigar essa ameaça, a arquitetura implementa autenticação multifator (MFA) e validação contínua de identidade, garantindo que apenas usuários autenticados possam acessar os recursos.
- **Interceptação de credenciais:** A interceptação de credenciais em trânsito é uma ameaça crítica em ambientes não-interativos. A arquitetura propõe a ofuscação de credenciais e o uso de HMAC para garantir a integridade das mensagens, dificultando a exploração por atacantes.
- **Acesso não autorizado ao Vault:** O acesso não autorizado ao cofre de senhas pode resultar em vazamento de segredos sensíveis. Para mitigar

esse risco, a arquitetura implementa políticas de acesso granular e rotação automática de credenciais, reduzindo a janela de exposição.

- **Injeção de SQL:** Ataques de injeção de SQL podem comprometer a integridade e confidencialidade dos dados armazenados no banco de dados. A arquitetura utiliza *prepared statements* e validação de entradas para prevenir esse tipo de ataque.

Essas estratégias garantem que a arquitetura proposta seja resiliente contra uma variedade de ameaças, proporcionando um ambiente seguro e confiável para a autenticação não-interativa e a gestão de segredos.

4.3 Validação do modelo e critérios de avaliação

A validação da arquitetura proposta é realizada por meio de testes práticos e simulações de ataques, acompanhadas de uma análise dos resultados, apresentada no Capítulo 6. O propósito é confirmar a eficácia do modelo *Zero Trust*, combinado com *IaC* e ofuscação de credenciais, na mitigação dos riscos de comprometimento de segurança.

Os critérios de avaliação incluem:

a) **Resiliência contra ataques:** impacto da ofuscação de credenciais e autenticação OTP/HMAC.
b) **Redução da exposição de segredos:** comparação com abordagens tradicionais.
c) **Conformidade com padrões de segurança:** aderência ao NIST SP 800-207 e CIS Controls v8.

Os testes de invasão incluem simulação de ataques de cibersegurança, como:

a) **Ataques de interceptação:** captura de credenciais durante a transmissão:
    1. Simulação da captura de credenciais em trânsito utilizando técnicas como *sniffing* de pacotes (ex.: Wireshark, tcpdump).
    2. Comparação entre a exposição de credenciais em texto claro e a versão ofuscada pelo modelo proposto.

b) **Ataques de força bruta:** tentativas de quebra de credenciais ofuscadas:
   1. Teste de tentativa de quebra de credenciais utilizando ataques força bruta sobre credenciais ofuscadas.

c) **Ataques *Insider*:** simulação de um agente interno comprometido:
   1. Simulação de um agente interno comprometido tentando obter credenciais sensíveis em arquivos de configuração, variáveis de ambiente e *logs* de sistema.

Para mensurar a efetividade do modelo, foi realizada uma comparação com arquiteturas tradicionais de autenticação, considerando os seguintes critérios, conforme apresentado no Quadro 6:

Quadro 6 – Comparação entre arquiteturas tradicionais e modelo proposto

Critério	Modelo Tradicional	Modelo Proposto
Armazenamento de credenciais	Senhas armazenadas em texto claro em arquivos de configuração ou banco de dados.	Credenciais ofuscadas dinamicamente antes do armazenamento e transmissão.
Proteção contra interceptação	Credenciais podem ser capturadas diretamente no tráfego de rede.	Ofuscação impede a recuperação direta das credenciais interceptadas.
Validação da autenticidade	Baseada apenas na correspondência estática da senha.	Autenticação dinâmica com OTP e validação por HMAC.
Exposição em logs e memória	Alto risco de exposição caso variáveis sejam registradas em arquivos temporários.	Ofuscação evita que as credenciais sejam legíveis diretamente na memória.
Resistência a ataques de repetição	Vulnerável caso a credencial seja reutilizada antes da expiração.	Tokens OTP são gerados dinamicamente e possuem validade limitada.
Facilidade de comprometimento	Se um atacante obtém a credencial, pode usá-la indefinidamente até ser alterada.	Mesmo que um atacante capture a credencial ofuscada, ele não pode usá-la sem o OTP correto.

Fonte: Do autor.

Os resultados e análise desses testes são apresentados na Seção 6.2 do Capítulo 6 (Avaliação e Análise de Resultados).

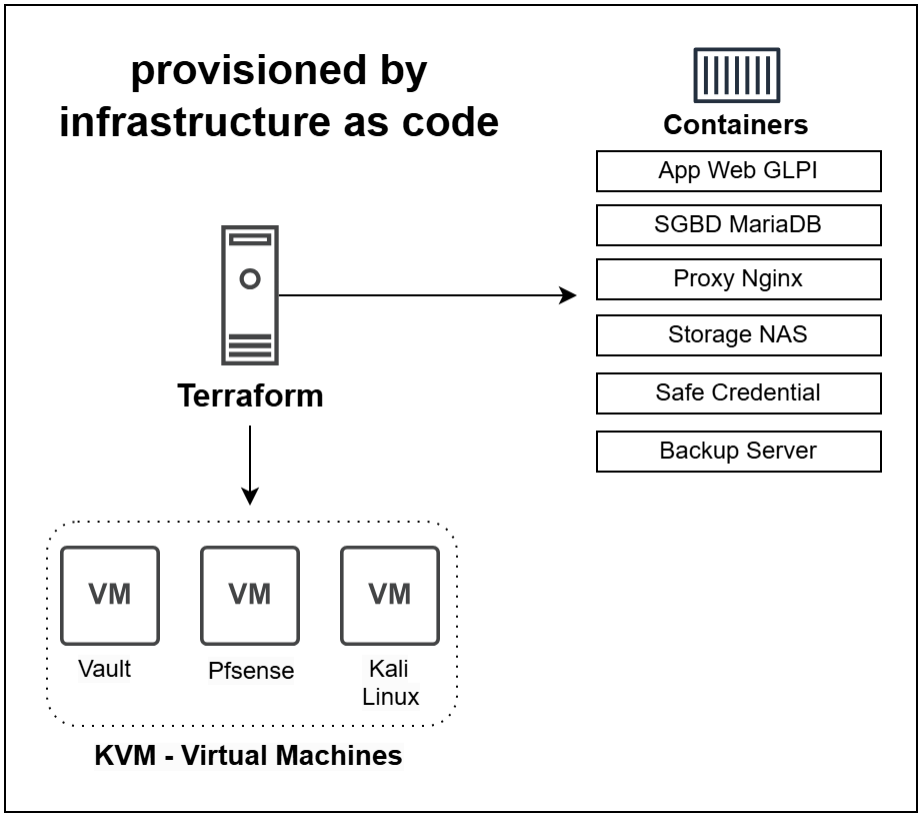
Capítulo 5

Protótipo

Este capítulo apresenta o protótipo desenvolvido durante a pesquisa, destacando os principais recursos, tecnologias e ferramentas que sustentam o modelo de arquitetura proposto. Entre os elementos centrais estão o mecanismo de proteção de credenciais, a integração com a arquitetura de rede ZTNA, o uso de um cofre de senhas e a aplicação de *IaC*.

A construção do protótipo foi realizada em um ambiente de laboratório e controlado com a utilização de diversas tecnologias para simular cenários reais. A Figura 16 ilustra uma visão geral dos recursos empregados no desenvolvimento do protótipo.

Figura 16 – Visão geral dos recursos utilizados no protótipo



Fonte: Do autor

5.1 Configuração do Ambiente

Na configuração do ambiente usou-se a ferramenta de *IaC* Terraform, com *containers* que instanciam diversos serviços e aplicações comuns em ambientes corporativos, utilizados na pesquisa.

O servidor principal, baseado no sistema operacional Linux – Ubuntu Server 22.04.5, foi configurado em uma máquina física. Este equipamento abriga uma série de aplicativos necessários para o funcionamento do protótipo, incluindo: HashiCorp Terraform e Vault, Docker, Twingate, KVM, PfSense e VSCode. Além disso, foram incorporadas ferramentas para validação estática de *IaC*, como Trivy, Checkov e Lynes, reforçando a segurança e a conformidade das configurações do ambiente.

O Quadro 7 descreve os serviços configurados:

Quadro 7 – Descrição dos serviços configurados no protótipo

Serviço	Descrição
Servidor App Web GLPI	Gerenciamento e manutenção de inventários de TI.
Servidor Banco de Dados MariaDB	Banco de dados integrado com o App Web GLPI.
Servidor Proxy Reverso Nginx	Gerencia o tráfego entre clientes e o servidor GLPI.
Servidor NAS	Simula um servidor de arquivos para armazenamento.
Servidor Backup	Responsável por armazenar cópias de segurança do ambiente.
CredentialSafe	Serviço de aplicação com o mecanismo de segurança implementado.

Fonte: Do autor.

Em todos os *containers*, foram instalados:

a) **Cliente Twingate:** Viabilizar a implementação da rede remota ZTNA, funcionando como um *proxy* local.
b) **Agente Vault (vault-ssh-helper):** Permite autenticação SSH integrada com o cofre de senhas.

Esses recursos facilitam a autenticação entre os componentes do mecanismo de segurança de credenciais e a comunicação segura com o cofre de senhas. As imagens dos

containers não foram reforçadas (sem *hardening*), permitindo demonstrar e aplicar as ferramentas de análise estática e vulnerabilidades, como:

a) **Trivy:** Verificação de vulnerabilidades.
b) **Checkov:** Identificação de configurações incorretas (*misconfigurations*) em *IaC*.
c) **Lynis:** Auditoria de segurança detalhada.

Além dos *containers*, três máquinas virtuais foram configuradas no equipamento principal para complementar o protótipo, conforme o Quadro 8:

Quadro 8 – Descrição das máquinas virtuais utilizadas no protótipo

Máquina Virtual	Descrição
Servidor Vault	Sistema operacional: Ubuntu Server 22.04. Executa o HashiCorp Vault, cliente Vault SSH, e cliente Twingate.
Servidor PfSense	Representa o *firewall* da arquitetura, com serviços de DNS e DHCP.
Desktop com Kali-Linux	Simula um cliente interno da rede para testes de segurança e invasão.

Fonte: Do autor.

No contexto do protótipo, o provedor de identidade (IdP) é considerado uma entidade externa ao ambiente, conforme detalhado no capítulo 4. A combinação das tecnologias empregadas resulta em uma solução integrada e segura, que possibilita o gerenciamento automatizado de credenciais em ambientes de *IaC*.

Essa abordagem está alinhada aos princípios do *Zero Trust*, permitindo mitigar os riscos inerentes às práticas tradicionais de segurança, como o uso de credenciais estáticas ou configuradas manualmente.

Para garantir a segurança e a conformidade do protótipo, foram planejadas as seguintes técnicas e ferramentas:

a) **Análise de código estático e dinâmico:** Avaliação do código para identificar potenciais vulnerabilidades antes e durante a execução.
b) **Exploração de vulnerabilidades conhecidas:** Testes para simular ataques com base em falhas documentadas em componentes do ambiente.
c) **Ferramentas automatizadas:** Uso de soluções especializadas para identificar falhas de segurança em *scripts* de *IaC* e configurações incorretas no ambiente.

Esses métodos serão descritos em detalhes na próxima subseção, destacando como contribuem para reforçar a robustez da solução.

No desenvolvimento do mecanismo de segurança, a linguagem de programação Python foi escolhida devido à sua versatilidade e ampla disponibilidade de bibliotecas voltadas para segurança. Entre as bibliotecas utilizadas, destacam-se:

a) **PyOTP**: para a implementação de OTPs (*One-Time Passwords*), garantindo autenticação dinâmica e baseada em tempo ou contador.
b) **HVAC:** para integração com o HashiCorp Vault, permitindo a gestão segura de segredos e credenciais.
c) **HMAC:** para assegurar a integridade e autenticidade das mensagens por meio de códigos de autenticação baseados em chave.

A integração do HVAC (*HashiCorp Vault API Client*) com o mecanismo proposto desempenha um papel fundamental na gestão segura de segredos e credenciais. O HVAC é uma biblioteca desenvolvida para facilitar a interação entre sistemas e o HashiCorp Vault, um cofre de senhas robusto que oferece armazenamento centralizado e seguro para segredos sensíveis, como chaves de API, senhas e *tokens*.

No contexto do mecanismo proposto, o HVAC possibilita:

a) **Automação do gerenciamento de credenciais:** O HVAC permite que o *NIA-Monitor* recupere credenciais de maneira programática e segura, garantindo que informações sensíveis não sejam expostas em texto claro em arquivos de configuração ou código.

b) **Geração e rotação de credenciais:** O HashiCorp Vault, por meio do HVAC, oferece a funcionalidade de rotação automática de credenciais, gerando novos segredos em intervalos regulares. Isso reduz significativamente os riscos de comprometimento por reutilização de credenciais.

c) **Segurança em camadas:** A integração com o Vault fortalece a abordagem de segurança em camadas ao garantir que todas as credenciais trafeguem de maneira segura e sejam armazenadas de forma criptografada no cofre.

d) **Ofuscação e desofuscação dinâmica:** As credenciais recuperadas do Vault pelo *NIA Monitor* passam pelo processo de ofuscação antes de serem enviadas ao *NIA Agent*. Isso garante que os segredos estejam protegidos tanto em trânsito quanto em repouso.

e) **Apoio à conformidade:** O uso do HVAC e do Vault assegura que o mecanismo proposto esteja em conformidade com melhores práticas de cibersegurança, como o CIS Controls v8, que recomenda a centralização da gestão de segredos e a adoção de mecanismos de acesso controlado.

f) **Flexibilidade e escalabilidade:** O HVAC facilita a integração do mecanismo de segurança com outros serviços e aplicações, permitindo que o sistema escale de forma eficiente em ambientes complexos e distribuídos.

O desenvolvimento da aplicação foi realizado utilizando o *Visual Studio Code* (VSCode), uma ferramenta robusta que oferece recursos avançados para edição de código, depuração e integração com controles de versão, facilitando o processo de implementação.

Para proteger os dados trafegados entre os componentes do sistema, foram adotados protocolos de segurança como HTTPS e TLS, que garantem a criptografia ponta a ponta das comunicações.

Além disso, a comunicação envolvendo o OTP foi implementada via OTP Socket, garantindo um canal seguro e confiável para a troca de informações sensíveis. Essa abordagem reforça a robustez do mecanismo, minimizando vulnerabilidades e assegurando a confidencialidade e integridade dos dados ao longo do processo.

5.2 Componentes tecnológicos da arquitetura

Na seção anterior, foi possível ter uma visão geral dos recursos presentes na arquitetura e a sua contribuição dentro do modelo proposto. Cada um desses recursos integra o ambiente proposto, permitindo uma abordagem integrada de segurança, dentro

de um padrão de arquitetura segura e moderna que, ao utilizar ZTNA (*Zero Trust Network Access*) e infraestrutura como código, oferece um ambiente computacional seguro e controlado.

Nas subseções seguintes, serão detalhadas as características técnicas de cada um desses componentes e como a sua combinação cria um ambiente operacional protegido e eficiente.

5.2.1 *Zero Trust Network Access* (ZTNA)

Para implementar os princípios de *Zero Trust* nesta arquitetura, escolheu-se a solução de ZTNA da Twingate, que permite construir um perímetro definido por *software*, gerenciando de forma centralizada o acesso de usuários e dispositivos a aplicações corporativas, sem necessidade de *hardware* externo ou mudanças na infraestrutura existente.

O Twingate destaca-se pela facilidade de implantação e por não demandar mudanças significativas na infraestrutura de rede existente, sendo uma opção eficiente para organizações que desejam implementar o modelo de segurança *Zero Trust*. Além disso, sua gratuidade para até 10 usuários foi um fator determinante na escolha desta solução para o trabalho.

Entre as alternativas possíveis de soluções ZTNA tem-se a Prisma *Access* (Palo Alto), Cloudflare *Access*, Citrix *Secure Private Access*. Todas elas são soluções coorporativas; existem outras disponíveis, mas não estudadas nesta pesquisa.

A solução de ZTNA desempenha um papel fundamental na arquitetura proposta. É útil comparar as *VPNs* tradicionais baseadas em perímetro com a abordagem de ZTNA para evidenciar as suas diferenças: enquanto as *VPNs* garantem apenas um perímetro de segurança genérico, o ZTNA proporciona um acesso mais seguro, segmentado e específico aos recursos corporativos, alinhando-se aos princípios de *Zero Trust*.

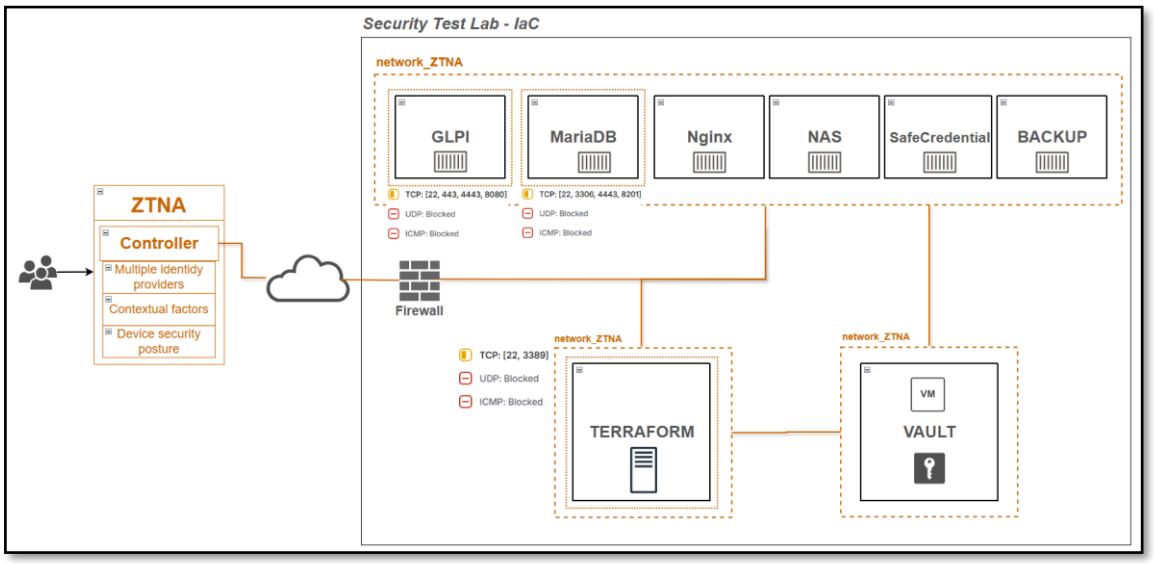
Compreender o funcionamento da arquitetura ZTNA é primordial para avaliar a aplicação desses elementos na arquitetura, sendo um aspecto fundamental a

microssegmentação baseada em recursos, que permite o isolamento lógico de cada recurso por *software*, o que facilita a gestão granular de acessos, portas e políticas.

Nesse contexto, a solução ZTNA permite ainda implementar o monitoramento contínuo das atividades para identificar padrões anômalos de acesso e ajustar as políticas de segurança em tempo real. Outra possibilidade é a adoção de políticas de acesso baseadas em identidade e contexto, que fortalece o controle sobre quem acessa o quê e quando, reduzindo riscos de movimentação lateral em caso de comprometimento.

A figura 17 ilustra os principais componentes ZTNA presentes na arquitetura. Por exemplo, o *contêiner* que hospeda a aplicação GLPI está segmentado de forma que apenas as portas TCP 22, 443, 4443 e 8080 estão liberadas, limitando o acesso a portas específicas para reduzir a superfície de ataque.

Figura 17 – Principais elementos ZTNA presentes na arquitetura proposta



Fonte: Do autor.

A segmentação granular demonstrada na figura 17 é orientada por recursos e controlada pelo *Controller* da solução, o que possibilita a aplicação de políticas de acesso detalhadas e uma visibilidade aprimorada do ambiente.

Por fim, a Figura 17 demonstra a aplicação do modelo de segurança em camadas alinhado ao conceito *Zero Trust*, onde:

a) **Segmentação de rede:** Os recursos estão segmentados na rede do Twingate *network_ZTNA*, onde o tráfego entre os recursos é controlado por políticas de *firewall* e pelo ZTNA *Controller*.

b) **Controle de acesso granular:** O ZTNA *Controller* gerencia quem pode acessar quais recursos e sob quais condições.
c) **Automação com *IaC*:** A integração do Terraform com o Vault e as políticas de segurança garante que a infraestrutura seja provisionada de maneira consistente e segura.

O Quadro 9 evidencia a proteção em camadas típica de uma abordagem *Zero Trust*, facilitando a explicação das barreiras de segurança aplicadas em cada etapa.

Quadro 9 – Visão geral das camadas *Zero Trust* da arquitetura proposta

Camada	Componentes
Autenticação	*ZTNA Controller, Identity Providers*
Rede	*Firewall, Nginx*
Aplicações	*GLPI, MariaDB, NAS, BACKUP*
Gerenciamento	*Terraform, Vault, SafeCredential*

Fonte: Do autor.

Em resumo, essa arquitetura demonstra uma implementação prática do modelo ZTNA, utilizando tecnologias como *firewall*, cofre de senhas, infraestrutura como código e segmentação de rede para proteger os recursos internos. O foco está na proteção contínua de identidades, dispositivos e recursos por meio de verificações constantes e controle de acesso baseado em políticas dinâmicas.

Essa abordagem reduz a superfície de ataque e garante que o acesso seja concedido apenas a usuários e dispositivos autenticados, minimizando os riscos de violações de segurança.

No protótipo, foi utilizado um servidor de identidades (IdP) padrão do mercado, o Google, com uma conta GMAIL. Esse IdP permite gerenciar de forma centralizada o acesso remoto a recursos, a configuração de ativos, bem como o gerenciamento de usuários, grupos e políticas de acesso através de uma interface *Web*. A ferramenta de ZTNA escolhida oferece acesso gratuito para até cinco usuários.

O acesso e a interação dos usuários com a ferramenta exigem autenticação obrigatória com o IdP, incluindo suporte para autenticação multifator (MFA) e controle de dispositivo para os sistemas operacionais iOS, Android e Windows. Isso permite

aplicar uma camada adicional de segurança, garantindo que apenas dispositivos autorizados possam acessar os recursos protegidos.

Além disso, toda a comunicação entre os recursos, ativos e clientes na rede criada pela solução ZTNA, chamada de "*network_ZTNA*", é protegida por meio da instalação de agentes ou aplicativos-clientes em todos os recursos. Esses agentes são configurados para garantir que o tráfego na *network_ZTNA* seja restrito e seguro.

Nesta implementação, utilizou-se uma conta de serviço (*service account*), que permite a autenticação automática e não-interativa dos recursos na plataforma ZTNA. Essa abordagem facilita a autenticação de recursos e a configuração automatizada, promovendo uma experiência de acesso seguro e simplificado sem exigir intervenção manual.

Essa configuração adere ao conceito de *Zero Trust*, onde cada acesso é verificado e controlado individualmente, reduzindo a exposição de recursos e aumentando a segurança da rede.

5.2.2 Infraestrutura como código (*IaC*)

O segundo componente da arquitetura é o uso de uma ferramenta para provisionamento e gerenciamento de *IaC*. No protótipo, utilizou-se o *software* Terraform da HashiCorp, que permite automatizar o provisionamento e a configuração de ambientes escaláveis e resilientes. O Terraform era considerado uma solução *open-source* até agosto de 2023, quando a empresa HashiCorp anunciou a mudança para a licença *Business Source License* (BSL), a partir da versão 1.6.0.

A BSL permite o uso gratuito do Terraform para finalidades não comerciais, com limites predefinidos para utilização. Embora o código do Terraform continue sendo público e auditável, ele deixou de ser completamente *open-source* devido às restrições comerciais introduzidas pela BSL.

Como alternativa ao Terraform, destaca-se o Pulumi, amplamente adotado pela comunidade *open-source* e reconhecido por sua abordagem moderna à automação de

infraestrutura. No entanto, a escolha do Terraform para esta pesquisa, iniciada no início de 2023, foi motivada por sua ampla utilização no mercado como o *software* de *IaC* mais consolidado e, à época, disponível gratuitamente.

A adoção do Terraform possibilita padronizar e versionar a infraestrutura, facilitando o controle de alterações e garantindo uma recuperação rápida em caso de falhas. Essa abordagem também contribui para a consistência nos ambientes e simplifica a replicação da infraestrutura em diferentes cenários (como desenvolvimento, teste e produção), reduzindo o risco de configurações manuais ou inconsistentes.

Além disso, a *IaC* auxilia diretamente na adoção dos princípios de *Zero Trust*, permitindo a implementação automatizada de políticas de segurança e segmentação, que são indispensáveis nessa abordagem.

Por exemplo, com a *IaC*, é possível definir regras de acesso granular para cada recurso, garantindo que somente usuários autenticados e autorizados possam acessar determinados serviços ou dados. Através do código, podem-se configurar controles que exigem autenticação e autorização contínuas, o que fortalece a postura de *Zero Trust* ao limitar o acesso a cada serviço e recurso de forma individual.

Neste cenário, a *IaC* permitiu a configuração de uma infraestrutura local, incluindo o provisionamento de *containers* para serviços como servidor *web* com aplicação, banco de dados, compartilhamento de arquivos, servidor de *backup*, *proxy* reverso e servidor *SafeCredential*. Em complemento, o Terraform facilita o provisionamento de máquinas virtuais e demais recursos de rede, ampliando a flexibilidade e escalabilidade da solução.

No código Terraform, foram adicionadas configurações para integrar as instâncias remotas e recursos *on-premises* com a solução de ZTNA, incluindo a definição dinâmica de *gateways* de acesso e políticas de roteamento. Essa integração permite que o controle de acesso e o gerenciamento de tráfego sejam centralizados, alinhando-se aos princípios de *Zero Trust*.

Por exemplo, as políticas podem ser ajustadas para garantir que cada interação com um recurso seja verificada por múltiplos fatores, como identidade do usuário, dispositivo e contexto.

O uso da *IaC* vai além do provisionamento, pois facilita a criação de *scripts* de configuração que suportam a implementação de políticas de segurança. Em particular, essas políticas garantem a conformidade com os princípios de *Zero Trust*, promovendo um acesso seguro e controlado aos recursos. Além disso, é possível manter registros e auditorias detalhadas de todas as alterações e acessos, o que permite a visibilidade completa do ambiente e facilita a identificação de atividades suspeitas.

Como parte dessas práticas de segurança, pode-se introduzir um programa eficaz de segurança de *containers*, focado em corrigir vulnerabilidades em tempo real e reduzir a superfície de ataque antes da implantação das imagens. Ao incorporar segurança na automação de tarefas e proteção da infraestrutura, garante-se que os *containers* sejam confiáveis e escaláveis.

Para identificar vulnerabilidades e configurações incorretas em *containers* e códigos *IaC*, utilizou-se no protótipo ferramentas de validação estática para código e imagens de *containers*, por exemplo: Trivy, Tfsec e Checkov. Essa estratégia é determinante para assegurar segurança e conformidade na infraestrutura provisionada. Os resultados dessa análise serão detalhados na seção de verificação de conformidade e análise estática, parte integrante do protótipo.

5.3 Implementação dos pressupostos de segurança no protótipo

O protótipo desenvolvido segue rigorosamente os pressupostos estabelecidos para garantir um ambiente seguro.

1. **Gestão segura de credenciais**
    - A autenticação de serviços é feita via Vault, eliminando a necessidade de armazenar segredos localmente.
    - A integração com OTP e HMAC impede ataques de repetição e falsificação de *tokens*.

2. **Aplicação de *Zero Trust* e microssegmentação**

   - O ambiente foi configurado para exigir autenticação forte em todas as interações.
   - O tráfego entre serviços é restringido por *firewalls* internos e controle de acesso granular.

3. **Segurança em infraestrutura como código**
   - Ferramentas como Checkov e Trivy foram utilizadas para varrer *scripts IaC* e identificar vulnerabilidades.
   - Configurações seguras foram aplicadas automaticamente, garantindo conformidade com os princípios *de Zero Trust*.

Dessa forma, o protótipo implementa as diretrizes de segurança estabelecidas, validando sua eficácia antes da análise final.

5.4 Implementação na redução da superfície de ataque

No protótipo, as estratégias para minimizar a superfície de ataque foram implementadas de forma prática, incluindo:

1. **Proteção de credenciais:**
   - Uso do Vault para armazenar segredos, eliminando a necessidade de credenciais em código.
   - Implementação de injeção dinâmica de segredos.

2. **Autenticação e controle de acesso:**
   - Uso de OTPs dinâmicos baseados em HOTP e HMAC.
   - Aplicação de *Zero Trust Network Access* (ZTNA) para limitar acessos não autorizados.

3. **Prevenção de ataques de repetição e engenharia reversa:**
   - Implementação de ofuscação de credenciais para dificultar extração.

- Validação de *tokens* antes de permitir acesso.

O uso dessas medidas fortalece a segurança do protótipo e reduz os riscos de exploração por atacantes.

5.5 Verificação de conformidade e segurança no ambiente proposto

No ambiente desenvolvido, a verificação de conformidade e segurança começa pela análise das configurações de *IaC,* um recurso importante para detectar parâmetros incorretos que possam expor dados e recursos críticos durante o provisionamento de redes e instâncias. Ferramentas como Trivy e Checkov permitem identificar vulnerabilidades e configurações inseguras em *scripts* de *IaC*, contribuindo para a proteção do ambiente antes da implantação.

O Trivy é uma ferramenta de análise estática que permite identificar vulnerabilidades em imagens de *containers*. Após o provisionamento dos *containers*, foi realizada uma varredura na imagem nginx:latest, na qual foram identificadas 95 vulnerabilidades de nível baixo, 34 de nível médio, 12 de nível alto e 2 críticas.

Esse tipo de análise permite aos operadores decidirem se irão provisionar a imagem vulnerável ou buscar alternativas mais seguras. O resultado da varredura apresenta um detalhamento das vulnerabilidades, utilizando o padrão CVE (*Common Vulnerabilities and Exposures*), que cataloga e identifica falhas com um CVE ID exclusivo (exemplo: CVE-2024-12345). Essa padronização, gerida pela MITRE Corporation, facilita a rastreabilidade e a divulgação de falhas, sendo amplamente adotada em ferramentas de segurança e relatórios de conformidade.

Além disso, a ferramenta oferece a possibilidade de realizar uma série de varreduras personalizadas, com diversas opções, como verificar exclusivamente CVEs classificadas como *CRITICAL*.

Outra ferramenta utilizada na verificação do protótipo é o Checkov, projetado para analisar códigos *IaC* em busca de configurações inseguras. A ferramenta realiza uma análise detalhada de diretórios que contenham arquivos com extensão .tf (Terraform),

com foco em detecção de senhas, credenciais ou *tokens* de acesso expostos no código, além da identificação de práticas inadequadas ou *security smells* em configurações *IaC*.

Os resultados obtidos com o Checkov validam a sua eficácia como uma ferramenta para proteger o ambiente antes da implantação.

No nível do sistema operacional, foi utilizada a ferramenta Lynis, especializada em auditorias de segurança para sistemas baseados em Linux, macOS ou Unix. A Lynis realiza verificações detalhadas para:

a) Identificar inconsistências no sistema operacional.
b) Testar conformidade com padrões internacionais, como GDPR, ISO27001, ISO27002 e PCI DSS.
c) Gerar recomendações para mitigação de riscos.

A ferramenta é customizável e amplamente utilizada em auditorias, testes de invasão, detecção de vulnerabilidades e *hardening* de sistemas. Os resultados dos testes de conformidade executados pela Lynis, que fornecem informações relevantes sobre a segurança do ambiente.

A aplicação de ferramentas como Trivy, Checkov e Lynis no ambiente do protótipo não apenas facilita a identificação de vulnerabilidades e configurações incorretas, mas também orienta na definição de ações práticas para mitigar riscos. Cada ferramenta oferece *insights* específicos sobre diferentes camadas da infraestrutura, abrangendo desde imagens de *containers* e configurações de *IaC* até o sistema operacional.

Com base nas descobertas realizadas por essas ferramentas, é possível implementar uma série de medidas corretivas e preventivas para fortalecer a segurança do ambiente. As ações descritas a seguir exemplificam como traduzir os resultados das análises em melhorias concretas, abordando as vulnerabilidades identificadas e promovendo a conformidade com as melhores práticas de segurança.

Além de corrigir problemas existentes, essas medidas contribuem para a construção de uma base mais robusta para futuras implantações, alinhando o ambiente aos princípios de segurança contínua e *Zero Trust*.

Os exemplos de ações de mitigação com base nas ferramentas são apresentados no Quadro 10.

Quadro 10 – Detecções e ações recomendadas após verificação

Trivy: Verificação de Vulnerabilidades em Imagens de *Contêiner*	
Detecções	**Ações recomendadas**
Pacotes ou bibliotecas desatualizados.	Identificar versões corrigidas (usando CVE IDs) e atualizar pacotes e bibliotecas vulneráveis.
Configurações padrão inseguras nas imagens (ex.: permissões excessivas).	Ajustar configurações para reduzir permissões ou adotar práticas de segurança padrão.
Componentes não utilizados aumentando a superfície de ataque.	Refatorar a imagem para remover componentes desnecessários.
Imagens vulneráveis usadas diretamente sem reforço de segurança (hardening).	Substituir por imagens certificadas ou aplicar *hardening* na imagem existente.
Checkov: Análise de Configurações em IaC	
Senhas, credenciais ou tokens expostos no código.	Remover segredos do código e armazená-los em um cofre seguro (ex.: HashiCorp Vault, AWS *Secrets Manager*).
Falta de encriptação em recursos sensíveis (ex.: volumes de armazenamento).	Configurar criptografia para dados em repouso e em trânsito.
Regras de segurança configuradas para acesso irrestrito (ex.: portas abertas ao público).	Restringir acesso a recursos com *firewalls*, listas de controle de acesso (ACLs) e regras de segurança refinadas.
Lynis: Auditoria de Segurança do Sistema Operacional	
Pacotes desatualizados ou vulneráveis.	Atualizar o sistema operacional e pacotes para versões corrigidas e compatíveis.
Serviços desnecessários executando com privilégios elevados.	Desativar ou remover serviços não essenciais para reduzir a superfície de ataque.
Configurações padrão inseguras em protocolos de rede ou autenticação (ex.: SSH).	Implementar hardening no SSH: usar autenticação por chave pública, desativar login *root* e ajustar algoritmos.

Fonte: Do autor.

A combinação de ferramentas como Trivy, Checkov e Lynis no contexto de um ambiente provisionado por *IaC* oferece uma abordagem eficiente para garantir segurança e conformidade. Entre as práticas recomendadas estão:

a) **Identificação de prioridades:** Classifique as vulnerabilidades por criticidade (baixa, média, alta) e priorize a resolução das mais críticas.
b) **Automatização de processos:** Configure *pipelines* de CI/CD para integrar ferramentas como Trivy e Checkov, interrompendo implantações caso vulnerabilidades críticas sejam detectadas.
c) **Aplicação de políticas de conformidade:** Baseie as configurações de segurança em padrões reconhecidos como ISO 27001 e PCI DSS, para assegurar a aderência às melhores práticas.

A análise de segurança realizada demonstrou a importância e a eficácia dessas ferramentas no contexto do protótipo. Mesmo com algumas vulnerabilidades ainda não corrigidas, foi possível validar os procedimentos, identificar configurações inseguras e garantir a conformidade do ambiente com as práticas de segurança. O uso dessas tecnologias reforça a robustez do modelo, permitindo a sua replicação em cenários reais.

5.6 Disponibilidade do código e repositório

Todos os procedimentos, códigos-fonte e implementações do protótipo estão disponíveis para consulta no repositório público, permitindo sua replicação total ou parcial. Essa transparência facilita a verificação dos resultados, a reutilização do modelo em outros projetos e a extensão para novas pesquisas ou aplicações práticas.

Além do código-fonte, o repositório inclui diagramas de sequência, imagens e demais documentações relevantes que detalham as etapas do desenvolvimento e os componentes do protótipo. Esses materiais complementares oferecem uma visão abrangente da pesquisa, possibilitando que outros pesquisadores ou profissionais compreendam e repliquem o modelo de forma mais eficiente.

Link do Repositório: GitHub - https://github.com/iacpucpr/jslangaro

Capítulo 6

Avaliação e Análise de Resultados

Este capítulo avalia a eficácia do modelo proposto, destacando as limitações das arquiteturas de segurança tradicionais baseadas em perímetro. Embora amplamente adotados, esses modelos se mostram insuficientes para atender às demandas de segurança em ambientes modernos, caracterizados pelo uso crescente de autenticação não-interativa e pela necessidade de uma gestão eficiente de segredos.

O objetivo é demonstrar como a combinação da arquitetura *Zero Trust* com *IaC* e cofres de senhas pode mitigar vulnerabilidades críticas e aprimorar a proteção de sistemas.

A avaliação é estruturada em três etapas principais: construção de um modelo de adversário e realização de testes de invasão (*pentest*) e análise de segurança. Essas etapas realizam a identificação de vulnerabilidades, o mapeamento de vetores de ataque, e a proposição de estratégias de mitigação adequadas, validando a eficácia da arquitetura.

A validação do modelo ocorre por meio de uma análise qualitativa dos resultados dos testes de invasão, que são orientados pelo modelo de adversário. Essa análise possibilita não apenas verificar a segurança da solução proposta, mas também identificar pontos críticos a serem aprimorados para fortalecer o modelo.

6.1 Modelo de adversário

O modelo de adversário desempenha um papel central na avaliação, mapeando ações típicas de atacantes, como escalada de privilégios e roubo de informações. Utilizando o *framework* Mitre ATT&CK (2024) foram identificadas táticas, técnicas e procedimentos (TTPs) baseados em ataques reais, permitindo uma análise detalhada das vulnerabilidades e das estratégias de mitigação aplicadas à arquitetura proposta.

O Quadro 11 sintetiza as táticas e técnicas aplicadas, os recursos comprometidos, os impactos observados e as mitigações sugeridas.

Quadro 11. TTP e mitigações da proposta

TÁTICA E TÉCNICA	RECURSO COMPROMETIDO	IMPACTO	MITIGAÇÃO
TA0001: Initial Access T1078: Valid Accounts	GLPI, MariaDB	O atacante pode acessar o banco de dados, através da credencial extraida do arquivo de configuração do GLPI.	Proteção imposta pela arquitetura ZTNA, com autenticação contínua para acesso aos recursos, e mecanismo de segurança proposto não expondo a credencial, além da utilização do cofre de senhas com a rotatividade de credenciais.
TA0004: Privilege Escalation T1068: Exploitation for Privilege Escalation	TERRAFORM	O atacante pode explorar vulnerabilidades de software, devido a configurações incorretas ou *security smells* em códigos *IaC* .	Proteção da arquitetura ZTNA com microsegmentação por recurso, ferramentas de análise estática para códigos *IaC* e imagens de *containers* , permitindo detectar configurações incorretas ou códigos inseguros.
TA0006: Credential Access T1552.001: Unsecured Credentials T1555.005: Credentials from Password Stores	GLPI, MariaDB, Nginx	O atacante pode acessar o banco de dados, através da credencial extraida de um arquivo em texto simples, ou arquivo de configuração ou exposta em código em uma API.	Credencial Ofuscada pelo mecanismo de segurança dificultará a ação do atacante, pois precisará fazer um *dump* de memória, depurar o programa, além das camadas impostas pela arquitetura ZTNA, que limitam a escalação de privilégios.
	VAULT, SafeCredential	O adversário pode conseguir acesso ao cofre de senhas ou ao mecanismo de proteção *SafeCredential* , e extrair credenciais. A depedência de um sistema externo para gerenciar senhas implica a necessidade de proteger esses sistemas de ataques.	Proteção imposta pela arquitetura ZTNA, restringindo a ação do atacante, pois somente administradores de sistemas podem ter acesso ao recurso. No modelo proposto, além dos administradores, somente o mecanismo proposto tem acesso ao cofre de senhas (Vault).
TA0008: Lateral Movement T1021.004 : Remote Services	Todos os recursos	O atacante pode explorar informações legítimas do usuário para acesso não autorizado à rede por meio do protocolo ssh.	Proteção imposta pela arquiteturza ZTNA, com autenticação contínua a cada recurso, microssegmentação por recurso, além do menor privilégio com acesso somente a recursos necessários. O uso do cofre de senhas é um elemento crucial no processo, tendo em vista a rotatividade imposta para a credencial, diminuindo assim a ação do atacante.

Fonte: Do autor.

No contexto da pesquisa, dois cenários de adversários foram considerados: um com acesso limitado aos recursos (por exemplo, servidores Nginx, GLPI e MariaDB) e outro com acesso total ao ambiente. A análise focou em três táticas específicas:

a) **Escalação de Privilégios (TA0004:T1068):** Adversários podem explorar vulnerabilidades no sistema ou configurações incorretas, como *security smells* em códigos *IaC*, para obter permissões elevadas. A mitigação inclui o

uso da arquitetura ZTNA para isolar recursos via microssegmentação e ferramentas de análise estática, como Checkov e Trivy, para identificar vulnerabilidades em códigos e *containers*.

b) **Credenciais Não Seguras (TA0006:T1552.001):** Adversários podem explorar credenciais armazenadas em texto simples ou incorporadas em código. A mitigação inclui o uso de cofre de senhas e a ofuscação de credenciais pelo mecanismo de segurança, que reduzem a exposição de dados sensíveis, dificultando a obtenção do valor real por meio de técnicas como *dump* de memória ou depuração de programas.

c) **Acesso a Serviços Remotos via SSH (TA0008:T1021.004):** Adversários podem explorar credenciais legítimas para obter acesso não autorizado. A mitigação envolve a aplicação de autenticação continua e uso do MFA, segmentação de rede e controle de acessos com base no princípio de menor privilégio.

Táticas adicionais, como acesso a credenciais e movimento lateral, foram tratadas com foco na combinação de práticas *Zero Trust* e validação contínua de configurações. O uso do HashiCorp Vault fortaleceu a gestão de credenciais, enquanto ferramentas como Lynis ajudaram a identificar vulnerabilidades antes de sua exploração.

Por fim, a abordagem *Zero Trust* complementa essas estratégias ao exigir verificação contínua de identidade, segmentação de rede e controle granular de permissões. Essa combinação resulta em ambientes mais resilientes, mitigando riscos associados ao uso indevido de credenciais e permitindo enfrentar adversários em cenários complexos e dinâmicos.

6.2 Teste de invasão ou *pentest*

Os testes de invasão foram conduzidos com base em metodologias reconhecidas internacionalmente, como PTES, OSSTMM, OWASP, PCI-DSS e as diretrizes do NIST, avaliando os vetores mais críticos, incluindo escalada de privilégios, movimento lateral e

exposição de credenciais. A solução ZTNA da Twingate foi aplicada para criar um ambiente controlado, limitando acessos remotos a endereços IP específicos.

Para garantir um cenário fiel às condições reais de segurança, não foram fornecidas senhas ou outros dados de acesso para realização dos testes.

A arquitetura avançada da Twingate possibilitou uma avaliação abrangente e controlada, permitindo a identificação de vulnerabilidades sem comprometer a integridade do ambiente.

Um dos destaques da solução foi sua capacidade de dificultar o reconhecimento de ativos, que é uma etapa comum nos testes de invasão − que geralmente envolve a coleta de informações sobre portas abertas e serviços em execução.

Devido a um recurso de segurança que apresenta todas as portas como abertas, o processo de reconhecimento tornou-se significativamente mais desafiador, reduzindo as chances de comprometimento rápido do sistema e evidenciando a robustez do modelo proposto. Os detalhes adicionais sobre o recurso de segurança podem ser encontrados no *link*:

- *https://help.twingate.com/hc/en-us/articles/4418888854417-TCP-IP-porttests-orscans-produce-inaccurate-results* (Twingate, 2024).

Os testes foram conduzidos com limitações predefinidas, como a exclusão de ataques de negação de serviço (DoS), engenharia social e técnicas de ataque tipo *Man-in-the-Middle* (MITM).

O método de teste *gray-box* foi adotado, caracterizado pelo acesso limitado do analista a informações do sistema, incluindo a rede Twingate e os ativos designados. Essa abordagem simula cenários reais, nos quais os invasores possuem algum grau de acesso ao ambiente, possibilitando uma avaliação mais fiel à realidade.

Para avaliar o risco de roubo de credenciais ou outras informações transmitidas na rede local, utilizou-se o TCPDump para coletar dados relacionados ao mecanismo de proteção, servidor *web* e banco de dados. Observa-se que os dados em trânsito estão protegidos por criptografia, seja por meio do protocolo HTTPS ou de conexões TLS com o soquete OTP, que torna o acesso às informações trafegadas na rede mais complexo.

Além disso, foi realizado um *dump* de memória para avaliar a possibilidade de acesso ao OTP armazenado. Embora viável, esse ataque requer privilégios administrativos elevados, e a instalação de bibliotecas específicas no sistema-alvo.

Contudo, o sucesso desse ataque depende da capacidade de capturar o conteúdo da memória no momento exato em que o HMAC está ativo, uma janela de oportunidade extremamente curta devido à validade limitada do HMAC.

No entanto, o cenário torna-se ainda mais restrito devido às validações contínuas e à microssegmentação impostas pela arquitetura de ZTNA, onde cada solicitação é verificada individualmente, limitando a exposição de dados sensíveis e isolando segmentos da rede, o que reduz drasticamente a superfície de ataque disponível para invasores.

A solução de ZTNA demonstrou robustez ao dificultar a coleta de informações sobre o ambiente testado. Isso reforça a eficácia do modelo proposto, que minimiza a superfície de ataque e protege informações sensíveis.

Outras medidas de segurança podem ser implementadas, como a configuração dos arquivos *hosts.deny* e *hosts.allow* para restringir acessos via SSH, bem como a aplicação de regras de localização no servidor *web* (Adaptado de Oliveira, *et al.*, 2024).

Além disso, a configuração de regras de *firewall* específicas e a adoção de ferramentas de proteção em nível de *host* são altamente recomendadas para reforçar a segurança. Essas práticas ajudam a minimizar a superfície de ataque, bloquear acessos não autorizados e proteger serviços críticos contra possíveis vulnerabilidades.

6.3 Análise de segurança

A análise de segurança comparou a arquitetura *Zero Trust* com o modelo tradicional baseado em perímetro. Conforme Oliveira *et al*. (2024), o modelo perimetral apresenta vulnerabilidades, como movimentos laterais e falhas de autenticação, que comprometem a eficácia das camadas de segurança interpostas.

No modelo *Zero Trust*, o acesso limitado à porta 443, protegido por autenticação multifator (MFA) e políticas de *Zero Trust Network Access* (ZTNA), reduz significativamente a superfície de ataque. Mesmo com credenciais comprometidas, um atacante precisaria realizar etapas complexas, como escalação de privilégios, *dumping* de memória e compreensão do algoritmo HOTP, além de decifrar o mecanismo de ofuscação.

Durante os testes de invasão, a varredura com NMAP revelou que todas as portas aparecem como abertas devido ao recurso de segurança da solução ZTNA. No entanto, outras ferramentas confirmaram que apenas as portas 443 (HTTPS) para conexões de internet, e as portas internas 5000 (mecanismo de segurança), 8201 (*key vault*) e 22 (SSH) estavam efetivamente liberadas. Essa configuração dificulta a ação de atacantes e demonstra a eficácia das medidas de segurança baseadas em *firewall*.

Para alcançar o objetivo do estudo, foi implementado um protocolo de autenticação que opera em paralelo com a abordagem tradicional, sem alterar o fluxo operacional dos mecanismos baseados em PAM (*Privileged Access Management*). Dessa forma, o tempo de validade das credenciais PAM, geralmente de 8 horas no mercado, permaneceu inalterado para acessos a bancos de dados.

No mecanismo proposto, mesmo que um invasor consiga roubar uma credencial ofuscada, será necessário depurar o programa para compreender a sua lógica, obter a sequência HOTP utilizada na ofuscação e reconstruir a semente empregada. Além disso, para acessar o banco de dados com a credencial original, o invasor precisará validar o código HMAC, gerado e verificado dinamicamente com base no OTP (*One-Time Password*) do momento. Como os códigos OTP e HMAC são renovados periodicamente, a autenticidade depende de sincronização temporal e de fatores adicionais, aumentando a complexidade para ataques.

O mecanismo de segurança desenvolvido adiciona uma camada extra de proteção ao ofuscar as credenciais, dificultando o acesso não autorizado mesmo em cenários onde a criptografia de transporte (como TLS/SSL) seja comprometida, como em ataques do tipo *Man-in-the-Middle*. Essa abordagem integrada reforça significativamente a proteção contra a interceptação de tráfego, pois, mesmo que os dados sejam capturados, a ofuscação impede a exposição direta de informações sensíveis.

Por fim, conforme os testes de invasão realizados por Oliveira *et al*., (2024), a implementação de ferramentas como Fail2ban ou soluções EDR (*Endpoint Detection and Response*) em servidores Linux ajuda a mitigar tentativas de força bruta em acessos SSH, complementando a estratégia de segurança proposta.

6.3.1 Validação dos pressupostos de segurança e impacto na redução da superfície de ataque

Para validar os pressupostos de segurança aplicados na arquitetura, foram conduzidos testes de invasão e análise de conformidade. Os resultados demonstraram que:

a) Ofuscação de credenciais e OTPs dinâmicos impediram a extração de segredos por meio de engenharia reversa.
b) A microssegmentação bloqueou tentativas de movimentação lateral, impedindo o acesso não autorizado a outros serviços.
c) O Vault garantiu que credenciais nunca estivessem expostas em código-fonte ou variáveis de ambiente, mitigando riscos de vazamento.
d) Uso de HOTP e proteção contra ataques de repetição evitou reutilização de *tokens* interceptados.

Os testes de segurança realizados demonstraram que a redução da superfície de ataque foi eficaz na mitigação de ameaças. O Quadro 12 traz uma comparação entre arquitetura tradicional e proposta.

Quadro 12 – Comparação com arquiteturas tradicionais

Critério	Arquitetura Tradicional	Arquitetura Proposta
Armazenamento de Credenciais	Senhas em código-fonte ou variáveis de ambiente	Cofre de senhas e injeção dinâmica
Proteção contra Engenharia Reversa	Senhas visíveis no código compilado	Ofuscação e OTPs dinâmicos
Controle de Acesso	Confiança implícita dentro da rede	*Zero Trust* e autenticação contínua

Prevenção de Movimentação Lateral	Rede plana, sem segmentação	Microssegmentação e *firewalls* internos

Fonte: Do Autor.

Os resultados demonstram que os pressupostos de segurança foram eficazes, reduzindo significativamente a superfície de ataque.

6.3.2 Comparação entre arquiteturas: Perímetro versus ZTNA

Ao contrário dos modelos tradicionais baseados em perímetro, o ZTNA adota um controle mais granular e dinâmico, utilizando microssegmentação e autenticação continua para minimizar riscos de movimento lateral e acessos não autorizados.

A seguir, destacam-se os principais aspectos que diferenciam essas abordagens, com foco no acesso granular, no controle baseado em identidade e contexto, e na visibilidade continua:

a) **Acesso granular e microssegmentação:** No modelo ZTNA, a microssegmentação permite isolar recursos específicos, como os containers GLPI, MariaDB e VM Vault, cada um com políticas de acesso detalhadas. Por exemplo, o container GLPI possui portas TCP específicas abertas (22, 443 e 5000), enquanto todo o tráfego UDP e ICMP é bloqueado. Essa segmentação reduz a superfície de ataque e dificulta o movimento lateral em caso de comprometimento.

   No modelo tradicional baseado em perímetro, os recursos compartilham um perímetro comum, tornando mais fácil para atacantes realizarem movimentos laterais e explorarem serviços disponíveis.

b) **Controle baseado em identidade e contexto:** O ZTNA avalia cada tentativa de acesso com base em múltiplos fatores, como identidade do usuário, localização, horário e postura de segurança do dispositivo. Com o uso de um controlador centralizado, é possível realizar ajustes dinâmicos e autenticações contínuas, evitando acessos irrestritos.

Já no modelo tradicional, a confiança é concedida após a autenticação inicial, permitindo acesso amplo aos recursos internos sem verificações adicionais de contexto. Essa abordagem aumenta os riscos de exploração, uma vez que não há reavaliações contínuas das permissões.

c) **Visibilidade e monitoramento contínuo:** O ZTNA proporciona visibilidade em tempo real de acessos e atividades, permitindo ajustes imediatos em políticas e bloqueios de acessos suspeitos. Essa capacidade facilita auditorias e a conformidade com normas de segurança.
Por outro lado, no modelo baseado em perímetro, o monitoramento é menos granular e frequentemente reativo, o que dificulta a detecção de acessos anômalos ou padrões suspeitos, comprometendo a capacidade de resposta a incidentes.

6.3.3 Exemplos técnicos e aplicações

A análise prática em laboratório destaca a aplicação do ZTNA para configurar a segmentação e o controle de acesso em serviços como o servidor APP GLPI e o Proxy Nginx, provisionados por meio de *IaC*. Esses serviços foram configurados com protocolos seguros (HTTPS) no lado do servidor antes da fase de produção, permitindo a validação dos componentes *Zero Trust* − uma tarefa que, tradicionalmente, seria realizada no lado do cliente.

A segmentação detalhada de recursos, como a abertura de portas específicas e o bloqueio de protocolos não utilizados, evidencia o controle granular proporcionado pelo ZTNA. Essa abordagem contrasta fortemente com a exposição típica observada em arquiteturas baseadas em perímetro, onde a falta de segmentação aumenta a superfície de ataque e os riscos de segurança.

6.3.4 Conclusão e relevância do modelo proposto

Com base na comparação entre as arquiteturas, conclui-se que a combinação do ZTNA com o *IaC* proporciona uma infraestrutura segura, resiliente e altamente adaptável. A integração de microssegmentação, controle granular baseado em identidade e automação de provisionamento reduz significativamente os riscos associados a movimentos laterais e acessos indevidos.

Além disso, essa abordagem simplifica a gestão e a escalabilidade da infraestrutura, garantindo conformidade com as políticas de segurança organizacionais. O uso do *IaC* automatiza não apenas o provisionamento, mas também a validação de componentes de segurança, assegurando que os serviços estejam alinhados aos princípios de *Zero Trust* ainda na fase da pré-produção.

Capítulo 7

Conclusão

Esta pesquisa apresentou uma arquitetura que integra tecnologias complementares, como *IaC*, princípios de *Zero Trust*, cofres de senhas e técnicas de ofuscação de credenciais. A combinação dessas soluções resultou em um modelo de segurança em camadas, projetado para mitigar vulnerabilidades críticas e proteger credenciais sensíveis em ambientes automatizados.

Os resultados dos testes de invasão (*pentest*) e da análise de segurança validaram a eficácia da arquitetura proposta, demonstrando sua capacidade de resistir a ataques que exploram credenciais expostas e configurações vulneráveis. Além disso, a integração de *Zero Trust* com *IaC* mostrou-se eficaz no gerenciamento seguro de credenciais, reduzindo a superfície de ataque e garantindo a escalabilidade do modelo.

No entanto, os testes também destacaram limitações nos métodos tradicionais de avaliação de segurança. Arquiteturas *Zero Trust*, que priorizam segmentação granular e autenticação contínua, exigem práticas de *pentest* adaptadas às suas especificidades. Essa constatação reforça a necessidade de desenvolver abordagens mais alinhadas a esse modelo, assegurando análises de segurança precisas e abrangentes em ambientes modernos.

A solução proposta promove ambientes computacionais resilientes, alinhados aos padrões atuais de segurança. Recomenda-se que as organizações mantenham um alinhamento contínuo entre tecnologia e processos, reforçando controles dinâmicos e proativos para acompanhar a evolução constante das ameaças de cibersegurança.

Por fim, é indispensável que profissionais de segurança e equipes técnicas adotem uma mentalidade de confiança mínima, princípio central do *Zero Trust*. A integração das

tecnologias abordadas demonstrou que uma abordagem multicamadas é eficaz para aumentar a segurança e a resiliência de sistemas.

O mecanismo de segurança proposto não apenas atendeu aos objetivos da pesquisa, mas também se mostrou viável e robusto, conforme comprovado pelos resultados dos testes de validação.

Referências

AHMADI, S. Zero Trust architecture in Cloud networks: Application, challenges and future opportunities. **Journal of Engineering Research and Reports**, [*s.l.*], v. 26, n. 2, p. 215-228, 2024. DOI: 10.9734/JERR/2024/v26i21083. Disponível em: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4725283. Acesso em: 14 out. 2024.

AHMED, A. **DevSecOps**: Enabling security by design in rapid software development. 2019. 113 f. Master's thesis (Business Informatics) – Utrecht University, Utrecht, The Netherlands, 2019. Disponível em: https://studenttheses.uu.nl/handle/20.500.12932/31896. Acesso em: 12 out. 2024.

ALJUNDI, M. K. **Tools and practices to enhance DevOps core values.** 2018. 78 f. Master's thesis (Computer Science) − Lappeenranta University of Technology, School of Business and Management, Lapperanta, Finland, 2018. Disponível em: https://lutpub.lut.fi/handle/10024/148944. Acesso em: 12 out. 2024.

ALONSO, J; PILISZEK, R; CANKAR, M. Embracing IaC Through the DevSecOps Philosophy: Concepts, challenges, and a reference framework. *In*: **IEEE Software**, Us/Canada, v. 40, n. 1, p. 56-62, Jan./Feb. 2023. Doi: 10.1109/MS.2022.3212194. Disponível em: https://ieeexplore.ieee.org/abstract/document/9915031. Acesso em: 15 set. 2024.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Microservices architecture enables DevOps: Migration to a cloud-native architecture. **IEEE Software**, Us/Canada, v. 33, n. 3, p. 42-52, May/June 2016. Disponível em: https://ieeexplore.ieee.org/abstract/document/7436659. Acesso em: 15 out. 2024.

BARACH, J. Towards Zero Trust Security in SDN: A multi-layered defense strategy. *In*: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING AND NETWORKING (ICDCN '25), 26., 2025, Hyderabad, Índia. **Proceedings […].** New York, NY, USA: Association for Computing Machinery, 2025. p. 331-339. Disponível em: https://dl.acm.org/doi/full/10.1145/3700838.3703671. Acesso em: 23 jan. 2025.

BASAK, S. K.; COX, J.; REAVES, B.; WILLIAMS, L. A comparative study of software secrets reporting by secret detection tools. *In*: ACM/IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT (ESEM), 2023, Nova Orleans, LA, EUA. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2023. p. 1-12. Disponível em: https://ieeexplore.ieee.org/abstract/document/10304853. Acesso em: 20 out. 2024.

BASAK, S. K.; NEIL, L.; REAVES, B.; WILLIAMS, L. What challenges do developers face about checked-in secrets in software artifacts? *In*: IEEE SECURE DEVELOPMENT CONFERENCE (SecDev), 2022, Atlanta, GA,

EUA. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2022. p. 69-76. Disponível em: https://ieeexplore.ieee.org/abstract/document/10172600. Acesso em: 14 out. 2024.

BEHERA, C. K; BHASKARI, D. L. Different obfuscation techniques for code protection. **Procedia Computer Science**, v. 70, p. 757-763, 2015. ISSN 1877-0509. DOI: 10.1016/j.procs.2015.10.114. Disponível em: https://www.sciencedirect.com/science/article/pii/S1877050915032780. Acesso em: 16 nov. 2024.

BLAZEJ, A. Secure and convenient secret management in distributed computer systems. **WSEAS Transactions on Computers**, New York, v. 15, 2016. E-ISSN: 2224-2872. Disponível em: https://wseas.com/journals/computers/2016/a565805-087.pdf#page=2.72. Acesso em: 16 nov. 2024.

BRITO, B. Segurança – Parte 1 – Como armazenar senhas. **AVERA**, [*s.l.*], 2019. Disponível em: https://www.brunobrito.net.br/seguranca-como-armazenar-senha/. Acesso em: dez. 2024.

BUCK, C.; OLENBERGER, C; SCHWEIZER, A; VÖLTER, F; EYMANN, T. Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust. **Computers & Security**, [*s.l.*], v. 110, 102436, 2021. ISSN 0167-4048. DOI: 10.1016/j.cose.2021.102436. Disponível em: https://www.sciencedirect.com/science/article/pii/S0167404821002601. Acesso em: 10 jan. 2025.

CAO, Y.; POKHREL, S. R.; ZHU, Y.; DOSS, R.; LI, G. Automation and orchestration of Zero Trust Architecture: Potential solutions and challenges. **Machine Intelligence Research**, [*s.l.*], v. 21, p. 294–317, 25 jan. 2024. Disponível em: https://link.springer.com/article/10.1007/s11633-023-1456-2?utm_source=chatgpt.com. Acesso em: 10 jan. 2025.

CHECK POINT. **What-is-infrastructure-as-code-iac?** [*S.l.*], c1994-2025a. Disponível em: https://www.checkpoint.com/pt/cyber-hub/cloud-security/what-is-infrastructure-as-code-iac/infrastructure-as-code-iac-security/. Acesso em: set. 2024.

CHECK POINT. **What-is-penetration-testing**. [*S.l.*], c1994-2025b. Disponível em: https://www.checkpoint.com/pt/cyber-hub/cyber-security/what-is-penetration-testing/. Acesso em: set. 2024.

CHICA, C. C. J.; IMBACHI, C. J; VEGA, B. F. J. Security in SDN: A comprehensive survey. **Journal of Network and Computer Applications**, v. 159, 102595, 2020. ISSN 1084-8045. DOI: 10.1016/j.jnca.2020.102595. Disponível em: https://www.sciencedirect.com/science/article/pii/S1084804520300692. Acesso em: 15 jan. 2025.

CIS CONTROLS. **CIS Critical Security Control 6**: Access Control Management. East Greenbush, NY, c2025. Disponível em: https://www.cisecurity.org/controls/access-control-management. Acesso em: 12 nov. 2024.

CISOFY. **Compliance solution**. The Netherlands, [2024?]. Disponível em: https://cisofy.com/compliance/. Acesso em: nov. 2024.

CODACY. Secrets-management: a complete guide. **Blog Codacy.com**, [*s.l.*], 22 nov. 2023. Disponível em: https://blog.codacy.com/secrets-management. Acesso em: 10 nov. 2024.

CONKLIN, L. Threat modeling process. **Owasp Foundation**, Wilmington, DE, [2024?]. Disponível em: https://owasp.org/www-community/Threat_Modeling_Process. Acesso em: 15 set. 2024.

CLOUDFLARE. **What-is-threat-modeling.** [*S.l*], c2025a. Disponível em: https://www.cloudflare.com/pt-br/learning/security/glossary/what-is-threat-modeling/. Acesso em: set. 2024.

CLOUDFLARE. **What-is-penetration-testing.** [*S.l*], c2025b. Disponível em: https://www.cloudflare.com/learning/security/glossary/what-is-penetration-testing/. Acesso em: set. 2024.

ERICH, F. DevOps is simply interaction between development and operations**.** *In:* **Lecture Notes in Computer Science** (including sub series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). LNCS, Springer International Publishing, v. 11350, 19 jan. 2019. Disponível em: https://link.springer.com/chapter/10.1007/978-3-030-06019-0_7. Acesso em: out. 2024.

FARROHA, B. S.; FARROHA, D. A framework for managing mission needs compliance and trust in the DevOps Environment. *In*: IEEE MILITARY COMMUNICATIONS CONFERENCE, out. 2014, Baltimore, MD. **Proceedings […].** Baltimore: IEEE, 2014. p. 288-293. Disponível em: https://ieeexplore.ieee.org/document/6956773. Acesso em: 15 out. 2024.

FORTINET. Zero Trust Security Model. **Fortinet.com**, [*s.l.*], c2025.Disponível em: https://www.fortinet.com/resources/cyberglossary/what-is-the-zero-trust-network-security-model?utm_source=blog&utm_campaign=zero-trust-network-security-model. Acesso em: 15 mar. 2024.

GARG. S; GARG. S. Automated cloud infrastructure, continuous integration and continuous delivery using Docker with robust container security. *In*: IEEE CONFERENCE ON MULTIMEDIA INFORMATION PROCESSING AND RETRIEVAL (MIPR), 2019, San Jose, CA, USA. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2019. p. 467-470. DOI: 10.1109/MIPR.2019.00094. Disponível em: https://ieeexplore.ieee.org/abstract/document/8695332. Acesso em: 15 mar. 2024.

GITHUB/PYAUTH. PyOTP - The Python One-Time Password Library. **GITHUB.com**, [s.l.], c2025. Disponível em: https://github.com/pyauth/pyotp. Acesso em: 12 ago. 2024.

GUERREIRO, M.; GARRIGA, D.; TAMBURRI, A.; PALOMBA, F. Adoption, support, and challenges of Infrastructure-as-Code: Insights from industry. *In*: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ICSME), 2019, Cleveland, OH, USA.

Proceedings […]. [*S.l.*]: IEEE Xplore, 2019. p. 580-589. DOI: 10.1109/ICSME.2019.00092. Disponível em: https://ieeexplore.ieee.org/document/8919181. Acesso em: 15 out. 2024.

GUO, X; XIAN, H; FENG, T; JIANG, Y; ZHANG, D; FANG, J. An intelligent zero trust secure framework for software defined networking. **PeerJ Computer Science**, v. 9, e1674, 2023. DOI: 10.7717/peerj-cs.1674. Disponível em: https://peerj.com/articles/cs-1674/. Acesso em: 15 out. 2024.

HÄKLI, A; TAIBI, D; SYSTA, K. Towards cloud native continuous delivery: An industrial experience report. *In*: IEEE/ACM INTERNATIONAL CONFERENCE ON UTILITY AND CLOUD COMPUTING COMPANION (UCC Companion), 2018, Zurich, Switzerland. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2018. p. 314-320. DOI: 10.1109/UCC-Companion.2018.00074. Disponível em: https://ieeexplore.ieee.org/abstract/document/8605798. Acesso em: 10 fev. 2024.

HASHICORP. What-is-Vault? **Hashicorp.com**, [*s.l.*], 2024. Disponível em: https://developer.hashicorp.com/vault/docs/what-is-vault. Acesso em: 15 set. 2024.

HOUSSAIN, H.; HOUSSAIN, A.; HOUSSAIN, Z.; SOHAG, H. I.; HAHMAN, S. OAuthSSO. A Framework to secure the OAuth-based SSO Service for packaged web applications. *In:* IEEE INTERNATIONAL CONFERENCE ON TRUST, SECURITY AND PRIVACY IN COMPUTING AND COMMUNICATIONS, 17., 2018, York, NY. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2018. p. 1575-1578. Disponível em: https://ieeexplore.ieee.org/document/8456096. Acesso em: 24 out. 2024.

HUMBLE, J.; FARLEY, D. **Continuous delivery**: Reliable software releases through build test and deployment automation. [*S.l.*]: Addison-Wesley Professional, 2010. Disponível em: https://ptgmedia.pearsoncmg.com/images/9780321601919/samplepages/0321601912.pdf. Acesso em: 24 out. 2024.

IBM. **Cost of a Data Breach Report 2023.** 2023a. Disponível em: https://www.ibm.com/reports/data-breach. Acesso em: 15 ago. 2024.

IBM. **A evolução do *Zero Trust* e os *frameworks* que o orientam**. 2023b. Disponível em: https://www.ibm.com/br-pt/think/insights/the-evolution-of-zero-trust-and-the-frameworks-that-guide-it? Acesso em: 15 dez. 2024.

IBM. **O que é IAM**? 22 jan. 2024. Disponível em: https://www.ibm.com/br-pt/topics/identity-access-management. Acesso em: 12 set. 2024.

IBRAHIM, A.; YOUSEF, A. H.; MEDHAT, W. DevSecOps: A security model for infrastructure as code over the cloud. *In*: International Mobile, Intelligent, And Ubiquitous Computing Conference (MIUCC), 2., 2022, Cairo, Egypt. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2022. p. 284-288. DOI: 10.1109/MIUCC55081.2022.9781709. Disponível em: https://ieeexplore.ieee.org/abstract/document/9781709. Acesso em: 10 out. 2024.

JIN, H. Research and practice of container system. *In:* INTERNATIONAL SYMPOSIUM ON THEORETICAL ASPECTS OF SOFTWARE ENGINEERING (TASE), 2021, Shanghai, China. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2021. p. 13-14. DOI: 10.1109/TASE52547.2021.00013. Disponível em: https://ieeexplore.ieee.org/abstract/document/9546738. Acesso em: 10 out. 2024.

KASPERSKY. **O que é um cofre de senha e para que serve**. c2025. Disponível em: https://www.kaspersky.com.br/resource-center/definitions/what-is-a-password-vault. Acesso em: set. 2024.

KINDERVAG, J. **No more chewy centers**: Introducing the Zero Trust model of information security. Forrester Research, 2010. Disponível em: https://media.paloaltonetworks.com/documents/Forrester-No-More-Chewy-Centers.pdf. Acesso em: 13 jan. 2025.

KHAIRUNISA, I; KABETTA, H. PHP source code protection using layout obfuscation and AES-256 Encryption Algorithm. *In*: INTERNATIONAL WORKSHOP ON BIG DATA AND INFORMATION SECURITY (IWBIS), 6., 2021, Depok, Indonesia. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2021. p. 133-138. DOI: 10.1109/IWBIS53353.2021.9631842. Disponível em: https://ieeexplore.ieee.org/abstract/document/9631842. Acesso em: 13 out. 2024.

KONALA, P. R.; KUMAR, V.; BAINBRIDGE, D. SOK: Static configuration analysis in infrastructure as code scripts. *In*: IEEE INTERNATIONAL CONFERENCE ON CYBER SECURITY AND RESILIENCE (CSR), 2023, Venice, Italy. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2023. p. 281-288. Disponível em: https://ieeexplore.ieee.org/document/10224925. Acesso em: 13 out. 2024.

KOSKINEN, A. **DevSecOps**: Building security into the core of DevOps. 2019. 64 f. Master's thesis (Information Systems) − University of Jyväskylä, Jyväskylä, Finland, 2019. Disponível em: https://jyx.jyu.fi/handle/123456789/67345. Acesso em: 13 out. 2024.

KRAWCZYK, H; BELLARE, M.; CANETTI, R. **HMAC**: Keyed-hashing for message authentication. Internet Engineering Task Force (IETF). Request for comments, 2104.1997. Disponível em: https://www.rfc-editor.org/rfc/rfc2104. Acesso em: 13 jan. 2025.

KREUTZ, D.; RAMOS, F. M. V.; VERISSIMO, P. E. *et al*. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, [*s.l.*], v. 103, n. 1, p. 14-76, 2015. DOI: 10.1109/JPROC.2014.2371999. Acesso em: 12 dez. 2024.

LOMBARDI, F.; FANTON, A. From DevOps to DevSecOps is not enough. CyberDevOps: an extreme shifting-left architecture to bring cybersecurity within software security lifecycle pipeline. **Software Quality Journal**, [*s.l.*], v. 31, p. 619-654, 26 abr. 2023. Disponível em: https://link.springer.com/article/10.1007/s11219-023-09619-3. Acesso em: 10 out. 2024.

LOHEST, C.; TOURPE, R. **Strengthening software security of Odoo**: An integrated approach using semgrep with rule-based modification and optimization. 2023. 79 f. Master's Thesis (Ingénieur Civil en Informatique, à finalité spécialisée) − Ecole Polytechnique de Louvain, Université Catholique de Louvain, Louvain, 2023. Disponível em: http://hdl.handle.net/2078.1/thesis:40053. Acesso em: 15 fev. 2024.

LWAKATARE, L. E.; KUVAJA, P.; OIVO, M. Relationship of DevOps to agile, lean and continuous deployment: A multivocal literature review study. *In*: INTERNATIONAL CONFERENCE PROFES, 17., 2016, Trondheim, Norway. **Proceedings of Product-Focused Software Process Improvement**. Lecture Notes in Computer Science LNCS, v. 10027. [*S.l.*]: Springer International Publishing, 2016. p. 399-415. Disponível em: https://link.springer.com/chapter/10.1007/978-3-319-49094-6_27. Acesso em: 13 out. 2024.

MARSHAL, A. Architecting network connectivity for a Zero Trust future. **Blog Network Connectivity Twingate**, [*s.l.*], 12 jan. 2022. Disponível em: https://www.twingate.com/blog/network-connectivity-for-zero-trust. Acesso em: 20 nov. 2024.

MACY, D. What is perimeter security in cybersecurity? **Security Forward**, [*s.l.*], 6 dez. 2023. Disponível em: https://www.securityforward.com/what-is-perimeter-security-in-cybersecurity/. Acesso em: 22 set. 2024.

MAZIERO, C. A. Sistemas operacionais: conceitos e mecanismos [recurso eletrônico]. Curitiba: DINF – UFPR, 2019. Disponível em: https://wiki.inf.ufpr.br/maziero/doku.php?id=socm:start. Acesso em: 22 set. 2024.

MEHRAJ. S.; BANDAY, M. T. Establishing a Zero Trust strategy in cloud computing environment. *In*: INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATION AND INFORMATICS (ICCCI), 2020, Coimbatore, India. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2020. p. 1-6. Disponível em: https://ieeexplore.ieee.org/abstract/document/9104214. Acesso em: 22 out. 2024.

MICUCCI, M. Ofuscação de código: uma arte que predomina na cibersegurança. **Welivesecurity By Eset**, [*s.l.*], 12 abr. 2024. Disponível em: https://www.welivesecurity.com/pt/recursos-e-ferramentas/ofuscacao-de-codigo-uma-arte-que-predomina-na-ciberseguranca/. Acesso em: dez. 2024.

MICROSOFT. **Zero Trust adoption report**: How does your organization compare? 2021. Disponível em: https://www.microsoft.com/en-us/security/blog/2021/07/28/zero-trust-adoption-report-how-does-your-organization-compare/. Acesso em: 13 mar. 2024.

MICROSOFT. **Zero Trust Security Model**. 2023. Disponível em: https://www.microsoft.com/security/zero-trust. Acesso em: 14 out. 2024.

MICROSOFT. **O que é gerenciamento de identidades e acesso (IAM)?** c2025. Disponível em: https://www.microsoft.com/pt-br/security/business/security-101/what-is-identity-access-management-iam. Acesso em: set. 2024.

MILLER, M. Secrets management overview & 7 best practices**. BeyondTrust**, [*s.l.*], 12 out. 2018. Disponível em: https://www.beyondtrust.com/blog/entry/secrets-management-overview-7-best-practices. Acesso em: 15 ago. 2023.

MITRE ATT@CK. **Mitre Att@ck Framework.** 2024. Disponível em: https://attack.mitre.org/. Acesso em: 12 set. 2024.

MOHAN, V.; BEN OTHMANE, L. SecDevOps: Is it a marketing buzzword? Mapping research on security in DevOps. *In:* INTERNATIONAL CONFERENCE ON AVAILABILITY, RELIABILITY AND SECURITY - ARES, 11., 2016, Salzburg. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2016. p. 542–547. Disponível em: https://ieeexplore.ieee.org/abstract/document/7784617. Acesso em: 10 out. 2024.

MORRIS, K. **Infrastructure as code**: Dynamic Systems for the Cloud. 2. ed. Boston: O'Reilly, 2016. 430 p. Disponível em: https://dl.ebooksworld.ir/books/Infrastructure.as.Code.2nd.Edition.Kief.Morris.OReilly.9781098114671.EBooksWorld.ir.pdf. Acesso em: 10 jan. 2024.

MYRBAKKEN, H.; COLOMO-PALACIOS, R. DevSecOps: A multivocal literature review**.** *In*: MAS, A.; MESQUIDA, A.; O'CONNOR, R.; ROUT, T.; DORLING, A. (eds). **Software process improvement and capability determination**. SPICE 2017. [*S.l.*]: Springer, Cham, 2017. (Communications in Computer and Information Science, v. 770). DOI: https://doi.org/10.1007/978-3-319-67383-7_2. Acesso em: 12 out. 2024.

MURIITHI, N. Infrastructure-as-code: Everything you need to know. **DZone Articles**, [*s.l.*], 27 mar. 2022. Disponível em: https://dzone.com/articles/infrastructure-as-code-everything-you-need-to-know. Acesso em: 14 fev. 2024.

NIST Special Publication 800-63B. **Digital Identity Guidelines**: authentication and lifecycle management. [*S.l.*]: National Institute of Standards and Technology (NIST), 2017/2020. Includes updates as of 03-02-2020, page VIII. Disponível em: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-63b.pdf. Acesso em: set. 2024.

NUNES, B. A. A.; MENDONCA, M.; NGUYEN, X. N. *et al*. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys & Tutorials,** [*s.l.*], v. 16, n. 3, p. 1617-1634, 2014. DOI: 10.1109/SURV.2014.012214.00180. Acesso em: 12 dez. 2024.

NYGARD, M. **Release It!** Design and deploy production-ready software. USA: Pragmatic Bookshelf, 2007. Disponível em: https://www.r-5.org/files/books/computers/dev-teams/production/Michael_Nygard-Design_and_Deploy_Production-Ready_Software-EN.pdf. Acesso em: 14 out. 2024.

OLIVEIRA, J. M.; OLIVEIRA, V. R.; MACEDO, D. F.; GUEDES, D.; NOGUEIRA, J. M. Abordagem confiança zero aplicada a ambientes computacionais big data: um estudo de caso. *In*: WORKSHOP DE GERÊNCIA

E OPERAÇÃO DE REDES E SERVIÇOS (WGRS), 27., 2022, Fortaleza. **Anais [...].** Porto Alegre: Sociedade Brasileira de Computação, 2022. p. 127-140. DOI: 10.5753/wgrs.2022.223549.Disponível em: www.sbc.com.br. Acesso em: 24 out. 2024.

OLIVEIRA, J.; SANTIN, A.; VIEGAS, E.; HORCHULHACK, P. A non-interactive one-time password-based method to enhance the Vault Security. *In*: BAROLLI, L. (ed.). **Advanced information networking and applications**. [*S. l.*]: Springer, 2024. p. 201-213. Disponível em: www.sbc.com.br. Acesso em: 24 out. 2024.

OMETOV, A.; BEZZATEEV, S.; MÄKITALO, N.; ANDREEV, S.; MIKKONEN, T.; KOUCHERYAVY, Y. Multi-factor authentication: A survey. **Cryptography**, [*s.l.*], v. 2, n. 1, 2018. DOI: 10.3390/cryptography201000. Disponível em: https://www.mdpi.com/2410-387X/2/1/1. Acesso em: 24 out. 2024.

OPDEBEECK, R; ZEROUALI, A; ROOVER, C. Control and data flow in security smell detection for infrastructure as code: Is It Worth the effort? *In*: IEEE/ACM INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES (MSR), 20., 2023, Melbourne, Australia. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2023. p 534-545. DOI: 10.1109/MSR59073.2023.00079. Disponível em: https://ieeexplore.ieee.org/document/10174011. Acesso em: 10 out. 2023.

PAHL, C; BROGI, A; SOLDANI, J.; JAMSHIDI, P. Cloud container technologies: A State-of-the-art review. **IEEE Transactions on Cloud Computing**, v. 7, n. 3, p. 677-692, July/Sept. 2019. DOI: 10.1109/TCC.2017.2702586. Disponível em: https://ieeexplore.ieee.org/abstract/document/7922500. Acesso em: 20 fev. 2024.

PEFFERS, K.; TUUNANEN, T.; ROTHENBERGER, M. A.; CHATTERJEE, S. A design science research methodology for information systems research**. J. Manag. Inf. Syst**., [*s.l.*], v. 24, n. 3, p. 45–77, 2007. DOI: 10.2753/MIS0742-1222240302. Disponível em: https://www.researchgate.net/publication/284503626. Acesso em: 14 out. 2024.

PETER, N. How to implement a Zero Trust security strategy. **Fortinet** [*s.l.*], 25 ago. 2021. Disponível em: https://www.fortinet.com/blog/industry-trends/how-to-implement-a-zero-trust-security-strategy. Acesso em: 14 mar. 2024.

PRAKASH, I. A. E. M. E. Unified secret management across cloud platforms: A strategy for secure credential storage and access. **Iaeme Publication**. 2024. DOI: 10.17605/OSF.IO/G96Z2. Disponível em: https://www.academia.edu/116287409/. Acesso em: 14 mar. 2024.

PRENEEL, B. Cryptographic hash functions. *In*: MAO, X. (ed.). **Advances in cryptology**. Berlin: Springer, 2019. p. 64-89. Disponível em: https://typeset.io/pdf/cryptographic-hash-functions-theory-and-practice-3do2bdrqz5.pdf. Acesso em: 10 jan. 2025.

QUEEN, C. What is infrastructure as code security? **Crowdstrike**, [*s.l.*], 14 maio 2024. Disponível em: https://www.crowdstrike.com/cybersecurity-101/cloud-security/iac-security/. Acesso em: 14 set. 2024.

RAHMAN, A.; WILLIAMS, L. Different kind of smells: Security smells in infrastructure as code scripts**. IEEE Security & Privacy**, [*s.l.*], v. 19, n. 3, p. 33-41, May/June 2021. DOI: 10.1109/MSEC.2021.3065190. Disponível em: https://ieeexplore.ieee.org/abstract/document/9388795. Acesso em: 24 out. 2024.

RAHMAN, A.; MAHDAVI-HEZAVEH, R.; WILLIAMS, L. A systematic mapping study of infrastructure as code research. **Information and Software Technology**, v. 108, p. 65-77, 2019. Disponível em: https://doi.org/10.1016/j.infsof.2018.12.004. Acesso em: 10 out. 2024.

RAHMAN, M. R.; IMTIAZ, N.; STOREY, M. A. *et al*. Why secret detection tools are not enough: It's not just about false positives - An industrial case study. **Empirical Software Engineering**, [*s.l.*], v. 27, p. 59, 2022. Disponível em: https://doi.org/10.1007/s10664-021-10109-y. Acesso em: 12 out. 2024.

RAHMAN, A. The seven sins: Security smells in infrastructure as code scripts. **Figshare Dataset**, 2019. Disponível em: https://ieeexplore.ieee.org/document/8812041. Acesso em: 12 out. 2024.

RAHMAN, A.; RAHMAN, M. R.; PARNIN, C.; WILLIAMS, L. Security smells in ansible and chef scripts: A replication study**. ACM Trans. Softw. Eng. Methodol**., [*s.l.*], v. 30, n. 1, 2021a. Disponível em: https://dl.acm.org/doi/abs/10.1145/3408897. Acesso em: 12 out. 2024.

RAHMAN, A.; BARSHA, F. L.; MORRISON, P. Shhh!: 12 practices for secret management in infrastructure as code. *In*: IEEE SECURE DEVELOPMENT CONFERENCE (SecDev), 2021b, Atlanta, GA, USA. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2021. p. 56-62. DOI: 10.1109/SecDev51306.2021.00024. Disponível em: https://ieeexplore.ieee.org/abstract/document/9652648. Acesso em: 14 out. 2024.

RED HAT. **O que há de diferente na segurança em nuvem?** 19 mar. 2018. Disponível em: https://www.redhat.com/pt-br/topics/security/cloud-security/. Acesso em: 24 mar. 2024.

RED HAT. **What is secrets management?** 14 maio 2024. Disponível em: https://www.redhat.com/en/topics/devops/what-is-secrets-management#:~:text=Secrets%20management%20is%20a%20method,DevOps%20or%20Security%20Ops%20workflows. Acesso em: set. 2024.

REED, P. J. **DevOps in practice.** 3. ed. Sebastopol, CA: O'Reilly, 2015. Disponível em: https://nmagee.github.io/cs4740/assets/oreilly-devops-in-practice.pdf. Acesso em: 22 jan. 2025.

RFC 6749. **The OAuth 2.0 Authorization Framework.** Out. 2012. Disponível em: https://www.rfc-editor.org/info/rfc6749. Acesso em: 14 set. 2024.

RIUNGU-KALLIOSAARI, L.; MÄKINEN, S.; LWAKATARE, L. E.; TIIHONEN, J.; MÄNNISTÖ, T. DevOps adoption benefits and challenges in practice: A case study. *In*: Abrahamsson, P. *et al*. (eds). **Product-Focused Software Process Improvement**. PROFES 2016. [*S.l.*]: Springer, Cham, 2016. (Lecture Notes in Computer Science, v. 10027). DOI: https://doi.org/10.1007/978-3-319-49094-6_44. Acesso em: 24 out. 2024.

RNP. **Gestão de identidade:** O que é e por que é importante? 2022. Disponível em: https://www.rnp.br/noticias/gestao-de-identidade-o-que-e-e-por-que-e-importante. Acesso em: 12 set. 2024.

ROADMAP.SH. **Cyber Security Expert**. 2024. Disponível em: https://roadmap.sh/cyber-security. Acesso em: 24 out. 2024.

ROBERT, L. Companies struggle with Zero Trust as attackers adapt to get around it. **DarkReading,** [*s.l.*], 26 jan. 2023. Disponível em: https://www.darkreading.com/remote-workforce/companies-struggle-zero-trust-attackers-adapt. Acesso em: 30 mar. 2024.

RONG, C.; GENG, J.; HACKER, T. J. *et al*. OpenIaC: Infraestrutura aberta como código – A rede é meu computador. **Journal of Cloud Computing**, [*s.l.*], v. 11, p. 12, 2022. Disponível em: https://doi.org/10.1186/s13677-022-00285-7. Acesso em: 14 out. 2024.

ROSE, S; BORCHET, O; MITCHELL, S; CONNELLY, S. **Zero Trust Architecture**. NIST Special Publication 800-207. Gaithersburg, MD: National Institute of Standards and Technology 2020. 59 pages CODEN: NSPUE2. DOI: https://doi.org/10.6028/NIST.SP.800-207. Disponível em: https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-207.pdf. Acesso em: 25 fev. 2024.

SAAVEDRA, N.; FERREIRA, J. F. Glitch: Automated polyglot security smell detection in infrastructure as code. **ArXiv,** Cornell University, CS, arXiv:2205.14371, 7 set. 2022. Parte superior do formulário

Disponível em: https://arxiv.org/abs/2205.14371. Acesso em: 12 out. 2024.

SCHNEIER, B. **Applied cryptography**: Protocols, algorithms, and source code in C. 20. ed. New York: John Wiley & Sons, 2015. 784 p. Disponível em: https://www.schneier.com/books/applied-cryptography/. Acesso em: 30 jan. 2025.

STALLINGS, W.; BROWN, L. **Computer security**: Principles and practice. [*S.l.*]: Pearson Book Digital Library, 2015. Disponível em: https://thuvienso.hoasen.edu.vn/handle/123456789/11970. Acesso em: 22 set. 2024.

SULTANA, M; HOSSAIN, A; LAILA, F; TAHER. K. A, ISLAM, M.N. Towards developing a secure medical image sharing system based on zero trust principles and blockchain technology. **BMC Med Inform Decis Mak**, [*s.l.*], v. 20, n. 256, 2020. Disponível em: https://doi.org/10.1186/s12911-020-01275-y. Acesso em: 20 set. 2024.

SYED, F. N; S. W. SHAH, W. S; SHAGHAGHI, A; ANWAR, A; BAIG, Z. AND DOSS, R. Zero Trust Architecture (ZTA): A comprehensive survey. **IEEE Access**, [*s.l.*], v. 10, p. 57143-57179, 2022. DOI: 10.1109/ACCESS.2022.3174679. Disponível em: https://ieeexplore.ieee.org/abstract/document/9773102. Acesso em: 20 set. 2023.

TWINGATE. What is Perimeter Security? **Blog Twingate**, [*s.l.*], 18 set. 2024. Disponível em: https://www.twingate.com/blog/glossary/perimeter-security. Acesso em: 24 set. 2024.

VITICCHIÉ, A; REGANO, L; TORCHINADO, M. BASILE, C; CECCATO, M; TONELLA, P. Assessment of source code obfuscation techniques. *In*: IEEE INTERNATIONAL WORKING CONFERENCE ON SOURCE CODE ANALYSIS AND MANIPULATION (SCAM), 16., 2016, Raleigh, NC, USA. **Proceedings […].** [*S.l.*]: IEEE Xplore, 2016. p. 11-20. DOI: 10.1109/SCAM.2016.17. Disponível em: https://ieeexplore.ieee.org/abstract/document/7781792. Acesso em: 22 set. 2024.

ZHANG, J. A Practical logic obfuscation technique for hardware security. **IEEE Transactions on Very Large-Scale Integration (VLSI) Systems**, [s.l.], v. 24, n. 3, p. 1193-1197, March 2016. DOI: 10.1109/TVLSI.2015.2437996. Disponível em: https://ieeexplore.ieee.org/abstract/document/7128395. Acesso em: 22 jan. 2025.

WANG, R. **Infrastructure as code patterns and practices**: With examples in Python and Terraform. Shelter Island, NY: Manning, 2022. (Book Collecion ITPro). Disponível em: https://books.google.com.br. Acesso em: 15 out. 2024.

WILSON, B. Infrastructure as code configuration management. DevOpsCube, [*s.l.*], 14 mar. 2023. Disponível em: https://devopscube.com/infrastructure-as-code-configuration-management. Acesso em: 10 out. 2024.