UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Lopes Ciccotti

Mecanismo de Acesso Remoto para Redes Privadas Utilizando Túnel SSH Reverso

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Lopes Ciccotti

Mecanismo de Acesso Remoto para Redes Privadas Utilizando Túnel SSH Reverso

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Diego Nunes Molinos

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2024

Matheus Lopes Ciccotti

Mecanismo de Acesso Remoto para Redes Privadas Utilizando Túnel SSH Reverso

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 22 de novembro de 2024:

Prof. Dr. Diego Nunes Molinos
FACOM/UFU
Orientador

Prof. Dr. Claudiney Ramos Tinoco
FACOM/UFU

Prof. Dr. Thiago Pirola Ribeiro
FACOM/UFU

Uberlândia, Brasil
2024

Resumo

O aumento do trabalho remoto e a crescente necessidade de acesso a redes privadas de forma segura trouxeram novos desafios para a conectividade em ambientes corporativos. Ferramentas de acesso remoto convencionais, embora úteis, enfrentam limitações em cenários onde altos níveis de segurança e eficiência são exigidos, especialmente em ambientes protegidos por *firewalls* e *CGNAT (Carrier-Grade Network Address Translation)*. Diante desse contexto, este trabalho propõe um mecanismo de acesso remoto utilizando túnel reverso SSH, integrando comunicação por meio de um *server broker*. A proposta visa facilitar a gestão de conexões remotas ao permitir o controle dinâmico das credenciais de acesso, elevando o nível de segurança e confiabilidade das conexões. A metodologia incluiu a análise de soluções existentes, desenvolvimento da arquitetura do sistema, e sua validação em um ambiente de teste real. Os resultados obtidos demonstraram que o mecanismo é capaz de contornar restrições de rede, garantindo acesso contínuo e seguro sem comprometer a integridade dos dados. Com essa abordagem, o trabalho contribui para o campo da segurança digital ao oferecer uma solução adaptada às necessidades de acesso remoto seguro em ambientes corporativos, ampliando as possibilidades de conexão em redes protegidas e elevando a segurança e eficiência das operações remotas.

Palavras-chave: Acesso remoto, SSH, Aplicação web, Segurança da informação, Túnel reverso, *Firewall*.

Abstract

The increase in remote work and the growing need for secure access to private networks have brought new challenges for connectivity in corporate environments. Conventional remote access tools, while functional, face limitations in scenarios where high levels of security and efficiency are required, especially in environments protected by firewalls and CGNAT (Carrier-Grade Network Address Translation). This work proposes a remote access mechanism using reverse SSH tunneling, integrating communication through a server broker. The proposal aims to facilitate the management of remote connections by allowing dynamic control of access credentials, enhancing the security and reliability of connections. The methodology included the analysis of existing solutions, system architecture development, and validation in a real test environment. The results showed that the mechanism can overcome network restrictions, ensuring continuous and secure access without compromising data integrity. This method enhances digital security by providing a tailored solution for secure remote access in corporate settings, expanding connectivity options within protected networks, and improving the security and efficiency of remote operations.

Keywords: Remote access, SSH, Web application, Information security, Reverse tunneling, Firewall.

Lista de ilustrações

Lista de abreviaturas e siglas

API	*Application Programming Interface*
BYOD	*Bring Your Own Device*
CGNAT	*Carrier-Grade Network Address Translation*
ER	Entidade-Relacionamento
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IP	*Internet Protocol*
MAC	*Media Access Control*
MQTT	*Message Queuing Telemetry Transport*
MVC	*Model-View-Controller*
NAT	*Network Address Translation*
NIST	*National Institute of Standards and Technology*
RDP	*Remote Desktop Protocol*
SHA	*Secure Hash Algorithm*
SSH	*Secure Shell*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UFW	*Uncomplicated Firewall*
UUID	*Universally Unique Identifier*
VPS	*Virtual Private Server*
VPN	*Virtual Private Network*
2FA	*Two-Factor Authentication*

Sumário

1 Introdução

Em 2022, conectaram-se à internet mais de 4,9 bilhões de pessoas, em 2023, até o mês de outubro, já haviam sido contabilizados aproximadamente 5,3 bilhões de pessoas conectadas, o que corresponde a 65,4% da população mundial (STATS, 2022; SHEWALE, 2023). De acordo com Nicoletti (2022), mesmo após anos da pandemia, inúmeras atividades estão retornando lentamente à normalidade, contudo, algumas mudanças são permanentes, dentre elas o trabalho remoto. Ainda de acordo com Nicoletti (2022), o trabalho remoto traz consigo inúmeros benefícios, tais como o aumento da produtividade e facilidade de comunicação, porém, esta transformação digital traz consigo desafios de segurança, acessibilidade e disponibilidade de recursos que antes estavam disponíveis no ambiente restrito da empresa.

Com o aumento do trabalho remoto, cresce também a necessidade de ferramentas de gestão de acesso seguro para as equipes de infraestrutura atuarem no alicerce tecnológico. Grande parte das ferramentas utilizadas para acesso remoto fazem uso do SSH *(Secure Shell)*, que desde sua concepção em 1995, tem experimentado uma evolução significativa no que tange a comunicação remota, principalmente em sistemas baseados em *UNIX*. Dentro deste contexto, técnicas mais atuais têm agregado segurança e dinamismo através da aplicação de túneis reversos SSH (SARAMAGO, 2020), permitindo conexões seguras mesmo em ambientes com restrições de acesso.

Apesar dos avanços proporcionados pelo SSH e pelas técnicas de túneis reversos, em muitos casos, especialmente em ambientes corporativos, ainda existem desafios para acessar sistemas legados, máquinas virtuais e aplicações web que operam na infraestrutura física de cada cliente (SARAMAGO, 2020). Isso ocorre porque nem sempre é possível ter acesso aos roteadores e *firewalls*[1] desses clientes, limitando a capacidade de realizar redirecionamentos de portas necessários. Geralmente, a autorização de acesso se restringe apenas aos servidores e seus recursos internos, sem permissão para alterar configurações de rede (MORENO, 2019).

Esta limitação cria desafios significativos para as equipes técnicas, que precisam gerenciar, atualizar ou solucionar problemas em sistemas críticos sem interromper a operação do cliente ou comprometer a segurança da rede. Diante disso, surge a necessidade de uma solução que permita o acesso remoto seguro aos serviços dos clientes, sem a necessidade de alterar configurações de *firewall* ou interferir na infraestrutura de rede existente.

Diante deste contexto, este trabalho propõe o desenvolvimento de um mecanismo

[1] Mecanismo de segurança que protege redes e computadores, controlando o tráfego de dados e bloqueando acessos não autorizados.

que utiliza túneis reversos SSH para conectar-se a máquinas clientes e, assim, possibilitar o acesso aos seus serviços. A arquitetura da ferramenta proposta permite que usuários autenticados abram e fechem túneis por meio de uma interface web, estabelecendo conexões seguras do cliente para o servidor. O sistema implementa medidas de segurança, como a geração de portas dinâmicas e a restrição de acesso baseada no IP externo do usuário, buscando proporcionar uma camada adicional de proteção na conexão remota.

1.1 Justificativa

Em ambientes corporativos, o acesso remoto a servidores e estações de trabalho é essencial para a manutenção, suporte e gestão contínua dos sistemas de informação. Contudo, a implementação desse acesso enfrenta desafios significativos relacionados à segurança e às restrições impostas pela infraestrutura de rede dos clientes. Uma prática comum para habilitar o acesso remoto é a abertura de portas específicas no *firewall*, como a porta 3389 utilizada pelo RDP (*Remote Desktop Protocol*)[2] (CLOUDFLARE, 2024). Entretanto, essa abordagem expõe os sistemas a riscos elevados de segurança, como ataques de força bruta e tentativas de invasão por *bots* maliciosos que exploram portas abertas para comprometer a integridade dos sistemas (MORENO, 2019).

Trabalhos como: Smith e Doe (2022), Lee e Chen (2021), Garcia e Patel (2023) demonstram que a abertura permanente de portas no *firewall* resulta em um aumento significativo de tentativas de acesso não autorizado, colocando em risco os ativos da organização e a confidencialidade das informações. Além disso, muitos clientes possuem políticas de segurança restritivas que impedem alterações na configuração de rede ou a abertura de novas portas, tornando inviável a utilização de soluções tradicionais de acesso remoto que dependem dessas modificações.

Ante o exposto o desenvolvimento de uma ferramenta que permita o acesso remoto seguro sem a necessidade de abrir portas no *firewall* ou modificar a infraestrutura de rede existente se faz necessária. A utilização de túneis reversos SSH surge como uma alternativa eficaz para contornar essas limitações, permitindo estabelecer conexões seguras iniciadas pelo cliente, que atravessam o *firewall* sem a necessidade de configurações adicionais. Essa abordagem diminui significativamente o risco de ataques de força bruta e varreduras de porta por agentes maliciosos (SMITH; DOE, 2022).

Adicionalmente, a capacidade de abrir e fechar túneis conforme a necessidade operacional, proporciona um controle granular sobre o acesso remoto, aumentando a segurança e a eficiência das operações de suporte e manutenção. Essa flexibilidade é particularmente vantajosa em ambientes onde o acesso remoto é esporádico ou deve ser ri-

[2] Protocolo proprietário desenvolvido pela Microsoft que fornece uma interface gráfica para conectar-se a outro computador pela rede.

gorosamente controlado, garantindo que os serviços estejam disponíveis somente quando necessário e sob supervisão adequada.

1.2 Objetivos

O objetivo deste trabalho é desenvolver um mecanismo de acesso remoto para conectar-se a máquinas clientes, permitindo o acesso aos seus serviços, mesmo sem a necessidade de alterar configurações no *firewall* ou utlização de *CGNAT (Carrier-Grade Network Address Translation)*[3].

Estabelecem-se como objetivos específicos: (a) Revisar os conceitos fundamentais de SSH e túneis reversos; (b) Compreender o modelo cliente/servidor no contexto de acesso remoto; (c) Apresentar a aplicação web desenvolvida para gerenciamento de túneis; descrevendo como permite que usuários autenticados abram e fechem conexões para acessar serviços de seus clientes; (d) Descrever as medidas de segurança implementadas na aplicação; (e) Demonstrar a solução proposta em um ambiente real.

[3] Tecnologia que permite que múltiplos usuários compartilhem um único endereço IP público, auxiliando na economia de endereços IPv4.

2 Fundamentação Teórica

Este capítulo apresenta os fundamentos teóricos que sustentam o trabalho. São abordados detalhadamente o protocolo SSH (Secure Shell), a técnica de túnel reverso SSH, o conceito de acesso remoto e a comunicação orientada a mensagem. Por fim, realiza-se uma breve análise dos trabalhos correlatos.

2.1 *Secure Shell (SSH)*

O protocolo SSH foi desenvolvido em 1995 por Tatu Ylönen, um pesquisador finlandês. O SSH, abreviação de *Secure Shell*, trata-se de um protocolo de rede que proporciona uma comunicação segura entre dispositivos em redes abertas. Diferenciando-se de protocolos mais antigos como *telnet* e rsh, o SSH introduz uma camada avançada de segurança, incluindo criptografia robusta e autenticação segura (BARRETT; SILVERMAN; BYRNES, 2005).

A versão original, conhecida como SSH-1, foi substituída pela SSH-2, que trouxe melhorias significativas em termos de segurança e funcionalidades (BARRETT; SILVERMAN; BYRNES, 2005). O SSH-2 resolveu várias limitações da versão anterior, introduzindo algoritmos de criptografia mais fortes, autenticação baseada em chave pública aprimorada e uma arquitetura mais modular (BARRETT; SILVERMAN; BYRNES, 2005).

O SSH é utilizado em uma variedade de cenários, tais como:

1. **Acesso Remoto:** Tornou-se a ferramenta padrão para gerenciamento remoto de sistemas *Unix* e *Linux*, além de equipamentos de rede, substituindo protocolos anteriores que não ofereciam a mesma segurança;

2. **Tunelamento:** O protocolo é eficaz na criação de túneis VPN (*Virtual Private Network*)[1], assegurando a proteção dos dados transmitidos entre dois pontos;

3. **Redirecionamento de Porta:** O SSH permite o acesso seguro a serviços em redes protegidas por *firewalls*, criando um canal seguro para a transmissão de dados.

Não obstante aos cenários acima, o SSH desempenha um papel crucial na segurança de sistemas automatizados e integrados. Conforme destacado por Ylonen et al. (2015), o SSH é frequentemente utilizado para acesso automatizado em uma variedade de propósitos, incluindo a gestão de grandes ambientes de TI, integração de aplicações

[1] Tecnologia que cria uma conexão segura e criptografada através de uma rede menos segura, como a internet.

e provisionamento de máquinas virtuais em serviços de nuvem. A flexibilidade e a segurança oferecidas tornam-no uma escolha ideal para essas operações críticas, onde a confidencialidade e a integridade dos dados são de suma importância.

2.2 Túnel Reverso

Túnel SSH é uma técnica de comunicação em redes que permite o transporte seguro de dados entre dois pontos em uma rede insegura, representando uma solução eficaz na proteção de informações sensíveis. Funcionando como uma *rede overlay* [2], o túnel SSH cria uma camada lógica adicional de comunicação sobre a rede existente, permitindo assim que os dados sejam transmitidos de forma segura e privada (SMITH; DOE, 2022).

O conceito de túnel SSH surgiu após a introdução do protocolo SSH no início dos anos 90. Desenvolvido originalmente como uma alternativa segura aos protocolos de rede existentes que não ofereciam criptografia, o SSH rapidamente se tornou um padrão para conexões seguras em redes inseguras. A capacidade de criar túneis surgiu como uma extensão natural do protocolo (DUSI; GRINGOLI; SALGARELLI, 2008).

O protocolo SSH, conforme descrito por Dusi, Gringoli e Salgarelli (2008), permite o túnel (redirecionamento de porta) de qualquer conexão TCP em cima do SSH, de modo a proteger criptograficamente qualquer aplicação que use protocolos em texto claro. De acordo com Hoffman (2019), uma variação do túnel SSH padrão é o túnel reverso, que permite "usar essa conexão estabelecida para configurar uma nova conexão do computador local de volta ao computador remoto". Nesse caso, o cliente estabelece a conexão inicial com o servidor e cria um canal seguro, permitindo que o servidor se conecte ao cliente. Essa técnica é especialmente útil em situações onde o cliente está atrás de um *firewall* e o redirecionamento de portas é inviável, sendo também aplicada em testes de desenvolvimento para expor servidores locais à internet.

Pode ser observado na Figura 1 um cenário clássico de túnel reverso. O servidor cliente, executando o RDP, está atrás de um roteador que realiza NAT, o que impede o acesso direto de usuários externos. No entanto, após a abertura de um túnel reverso do servidor cliente para um servidor remoto acessível a todos, realizando o encaminhamento das requisições RDP da porta 40000 para a porta 3389, permite-se que os usuários, ao acessarem a porta 40000 desse servidor remoto, possam utilizar o RDP do cliente que está atrás do *firewall*.

[2] Rede virtual criada sobre uma infraestrutura física existente, utilizando mecanismos de encapsulamento para conectar dispositivos ou sistemas de forma lógica, independente da topologia física subjacente.

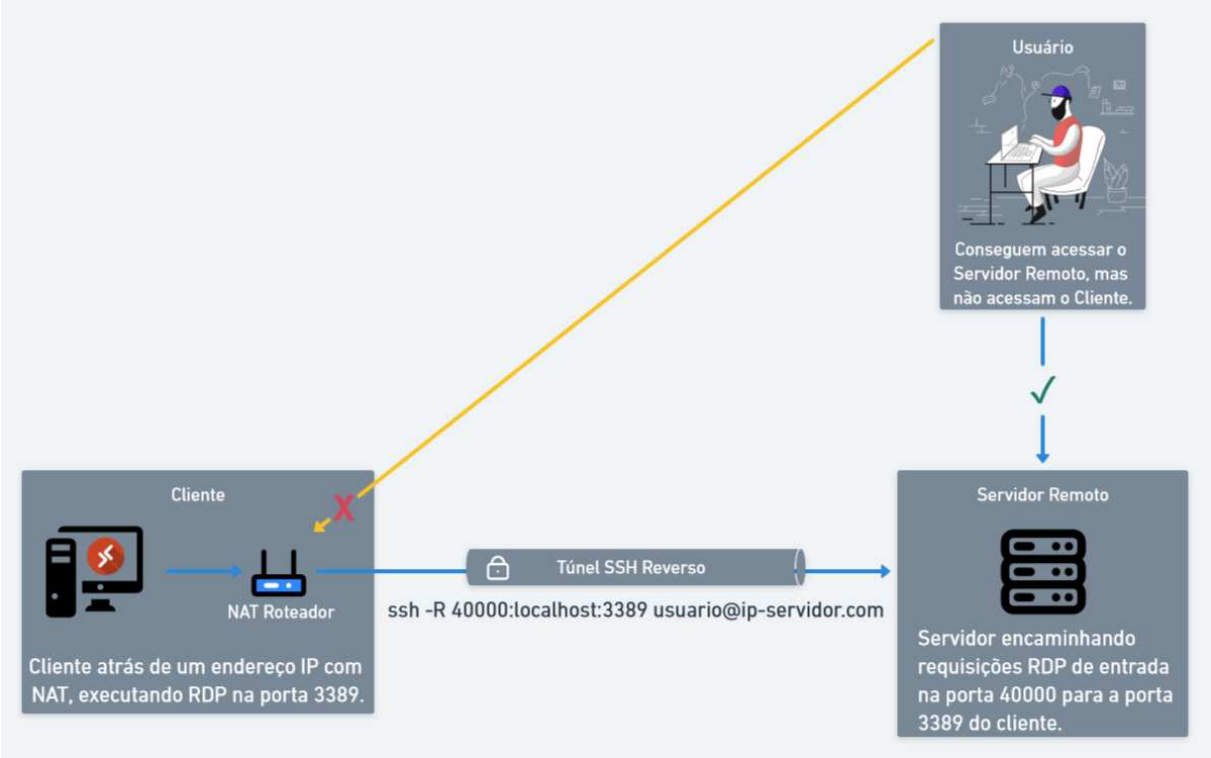

Figura 1 – Mecanismo de Túnel SSH Reverso. Figura adaptada de <https://goteleport.com/blog/ssh-tunneling-explained/>.

À medida que a segurança cibernética evolui, é provável que os túneis SSH continuem a desempenhar um papel vital no transporte seguro de dados sensíveis, adaptando-se às novas ameaças e desafios para garantir a integridade e a confidencialidade das informações em um mundo cada vez mais conectado.

2.3 Acesso Remoto

O acesso remoto permite que usuários de uma organização acessem recursos computacionais de locais externos, sem estar nas instalações da organização. Essa tecnologia, que se tornou ainda mais relevante com o aumento do *home office*, utiliza dispositivos variados como computadores, *smartphones* e *tablets* para realizar uma gama de tarefas (OLIVEIRA; SOUZA, 2021).

De acordo com Nogueira e Patini (2012), as soluções modernas de acesso remoto são projetadas para garantir a confidencialidade, integridade e disponibilidade dos dados, o que ocasiona diversos desafios de segurança para as equipes de tecnologia. Como o *National Institute of Standards and Technology* (NIST) destaca, "Os objetivos de segurança mais comuns para tecnologias de teletrabalho e acesso remoto são: confidencialidade - garantir que as comunicações de acesso remoto e os dados do usuário armazenados não possam ser lidos por partes não autorizadas; integridade - detectar quaisquer alterações

intencionais ou não intencionais nas comunicações de acesso remoto que ocorram durante a transmissão; e disponibilidade - garantir que os usuários possam acessar recursos por meio do acesso remoto sempre que necessário" (SOUPPAYA; SCARFONE, 2016).

Além disso, o acesso remoto tem se adaptado para suportar uma variedade de dispositivos e plataformas, incluindo aqueles no modelo BYOD - *Bring Your Own Device*, onde os usuários utilizam seus próprios dispositivos para acessar recursos da organização. Essa flexibilidade, no entanto, traz desafios adicionais de segurança. O NIST adverte que as "organizações devem planejar sua segurança de acesso remoto assumindo que as redes entre o dispositivo de teletrabalho e a organização não podem ser confiáveis" (SOUPPAYA; SCARFONE, 2016).

Exemplos práticos incluem o uso de VPN para conectar funcionários remotos de forma segura às redes corporativas, o uso de softwares de *desktop* remoto que permitem aos usuários controlar um computador à distância como se estivessem fisicamente presentes, e a utilização do protocolo SSH, que oferece um método seguro de acesso remoto a um servidor por meio de um túnel criptografado, garantindo a segurança na transmissão de dados.

O acesso remoto vai além de uma mera ferramenta tecnológica, sua relevância se destacou especialmente durante a pandemia causada pela COVID-19, onde provou ser crucial para a manutenção das atividades empresariais e educacionais (SILVA; SANTOS, 2020). Olhando para o futuro, é evidente que o acesso remoto continuará a ser essencial na era digital, influenciando a maneira como interagimos, trabalhamos e aprendemos em um mundo cada vez mais conectado.

2.4 Trabalhos Relacionados

Nesta seção, são apresentados trabalhos que possuem relação com os objetivos deste projeto. Ora os trabalhos abordam parte da técnica utilizada de SSH reverso, ora abordam a aplicação da técnica como estudo de caso em cenários já conhecidos.

A Tabela 1 destaca os trabalhos analisados sob a óptica do tipo de publicação e o Qualis do referido periódico ou evento.

Os trabalhos analisados apresentam propostas relevantes para a aplicação de túneis SSH reversos. Uma proposta de software para estabelecimento de conexões seguras via *Shell* é apresentada em Toledo (2023). Os autores discutem a dificuldade de acessar computadores remotos para realização de tarefas gerais de informática, tais como manutenção, configuração ou até mesmo depuração. A ferramenta apresentada aborda os desafios impostos por barreiras como *firewalls*, roteadores e inúmeras configurações, que tornam o processo complexo para usuários comuns. Caracteriza-se pela facilidade de uso,

Tabela 1 – Trabalhos Relacionados

Título	**Autores**	**Resumo**
Desenvolvimento de uma aplicação Web para acesso remoto a computadores de bordo utilizados no ramo da automação agrícola	*Maria Victória Mundim Saramago*	Proposta de uma aplicação web para acessar computadores de bordo na automação agrícola.
SmartSockets: Solving the Connectivity Problems in Grid Computing	*Jason Maassen; Henri E. Bal*	Biblioteca para resolver problemas de conectividade em grids usando sockets inteligentes.
Research on Video Surveillance Robot Based on SSH Reverse Tunnel Technology	*Xibin Pan; Hongming Hu; Jianping Xu; Mangyan Li*	Uso de túneis SSH reversos para comunicação de robôs de vigilância.
SSH tunneling to connect to remote computers	*Sivan Toledo*	Discussão sobre o uso de túneis SSH para conexão segura a computadores remotos.
rvGAHP: push-based job submission using reverse SSH connections	*Scott Callaghan; Gideon Juve; Karan Vahi; Philip J. Maechling; Thomas H. Jordan; Ewa Deelman*	Proposta de um sistema para submissão de tarefas usando conexões SSH reversas.

baixo número de dependências, muitas garantias de segurança e escalabilidade para gerenciar grandes quantidades de computadores remotos. Torna-se importante mencionar, entretanto, que a solução proposta possui dependência direta de alguns serviços de nuvem pagos.

A ferramenta proposta em Toledo (2023) utiliza a técnica de SSH reverso para superar as dificuldades de configuração na rede do cliente. No entanto, ela depende do protocolo MQTT [3] (HUNKELER; TRUONG; STANFORD-CLARK, 2008), que não foi projetado para oferecer os níveis de segurança necessários no fornecimento das credenciais de acesso, as quais são gerenciadas por meio de um servidor *Broker*[4]. Além disso, a ferramenta utiliza a plataforma pública e gratuita *GitHub* para o compartilhamento de chaves de segurança entre cliente e servidor, o que pode apresentar riscos adicionais. De acordo com os autores, apesar dessas limitações, a solução atende aos requisitos mínimos de conexão remota via SSH reverso e, quando comparada a ferramentas similares, apresenta um baixo custo operacional.

Em Callaghan et al. (2017), a técnica de SSH reverso é praticada com o objetivo de alcançar *performance* computacional em relação ao processamento de tarefas em grande escala. As ferramentas utilizadas para processamento de alto desempenho em ambientes distribuídos, heterogêneos, normalmente dependem de uma abordagem baseada em *Push/Pull* para provisionamento de recursos de processamento. Contudo, muitos desses sistemas, tais como *Clusters*, têm utilizado mecanismos eficientes de autenticação, como

[3] *Message Queuing Telemetry Transport*, protocolo de mensagens leve para sensores e dispositivos móveis otimizado para redes de baixa largura de banda.

[4] Intermediário que gerencia a comunicação entre dispositivos que publicam e assinam mensagens no protocolo MQTT.

por exemplo: autenticação em dois fatores, o que apresenta um nível maior de dificuldade em transmitir cargas de processamento baseadas em *Push/Pull*. Diante deste contexto, Callaghan et al. (2017) descreve uma abordagem que permite enviar *Jobs* para serem processados por um sistema de alta *performance* utilizando SSH reverso. De acordo com os autores, o método apresenta resultados mais eficientes em relação ao método *Push/Pull* tradicional.

No estudo realizado por Pan et al. (2020), a técnica de túnel reverso SSH é empregada para resolver problemas de conectividade em sistemas de vigilância. Nesse trabalho, os autores propõem o desenvolvimento de um robô de vigilância com capacidade de controle remoto e transmissão de vídeo em tempo real. A solução utiliza um servidor em nuvem como intermediário para conectar um terminal móvel e o robô, superando barreiras de NAT e *firewalls* que impedem a comunicação direta. A integração de um *Raspberry Pi* e um *Arduino* com a técnica de túnel reverso permite controlar o movimento do robô e capturar vídeos em tempo real, os quais são transmitidos ao terminal móvel. Essa abordagem demonstra aplicabilidade em segurança residencial, automação doméstica e resgate em desastres.

Já a biblioteca *SmartSockets*, introduzida em Maassen e Bal (2007), oferece uma solução para os desafios de conectividade em ambientes de *grid computing*, especialmente em ambientes com *Firewall*, NAT (*Network Address Translation*) e complexidades de *Multi-Homing*. Esse trabalho se destaca pela implementação de quatro mecanismos distintos de configuração de conexão: Direta, Reversa, *Splicing* e Roteada, onde cada um desses métodos foi projetado para diferentes tipos de restrições de rede, proporcionando assim uma solução abrangente para vários cenários de conectividade. A *SmartSockets* é caracterizada pela facilidade de uso, atuando automaticamente em uma ampla gama de problemas de conectividade, com mínima intervenção do usuário. De acordo com os autores, a *SmartSockets* integra soluções conhecidas como encaminhamento de porta e tunelamento SSH, bem como abordagens para desafios de *Multi-Homing* e identificação de máquinas. Conforme apresentado no trabalho, os testes realizados em seis sites diferentes ao redor do mundo, a *SmartSockets* conseguiu estabelecer conexões bem-sucedidas em todas as 30 tentativas, demonstrando uma eficácia significativamente maior em comparação com as conexões convencionais, que tiveram sucesso em apenas seis tentativas.

Em Saramago (2020) é detalhado o desenvolvimento de uma solução para acesso remoto integrada a sistemas embarcados para uso em máquinas agrícolas, com foco na auditoria de sessões e controle de acesso. A solução apresentada faz uso de redirecionamento de mensagens para o *WebSocket*, estabelecendo a conexão através de um módulo embarcado. O método SSH reverso é empregado para estabelecer túneis SSH entre o computador do usuário e o servidor intermediário. O sistema verifica a compatibilidade da versão de software do *display* antes de iniciar uma sessão de acesso remoto, garantindo a

validade da conexão com base na associação entre o *display* e o usuário.

Por fim, as diferentes abordagens estudadas nos trabalhos relacionados oferecem *insights* valiosos para o desenvolvimento de sistemas de acesso remoto. Especialmente, a utilização do SSH reverso emerge como uma solução comum para contornar restrições de rede, embora ainda existam lacunas em termos de segurança e detalhamento técnico.

3 Método

Segundo as classificações de Wazlawick (2009), este trabalho possui caráter exploratório e experimental, seguindo a vertente de propor algo "presumivelmente melhor" em relação ao que já existe. É fundamental que o objetivo do trabalho esteja bem definido para orientar as decisões metodológicas. Assim, dado que este trabalho visa desenvolver um mecanismo de acesso remoto que utiliza túnel reverso SSH para ambientes corporativos, a metodologia contempla as etapas de especificação, desenvolvimento e validação, bem como uma etapa de análise e resultados.

A Figura 2 ilustra, do ponto de vista metodológico, todas as etapas da construção da solução.

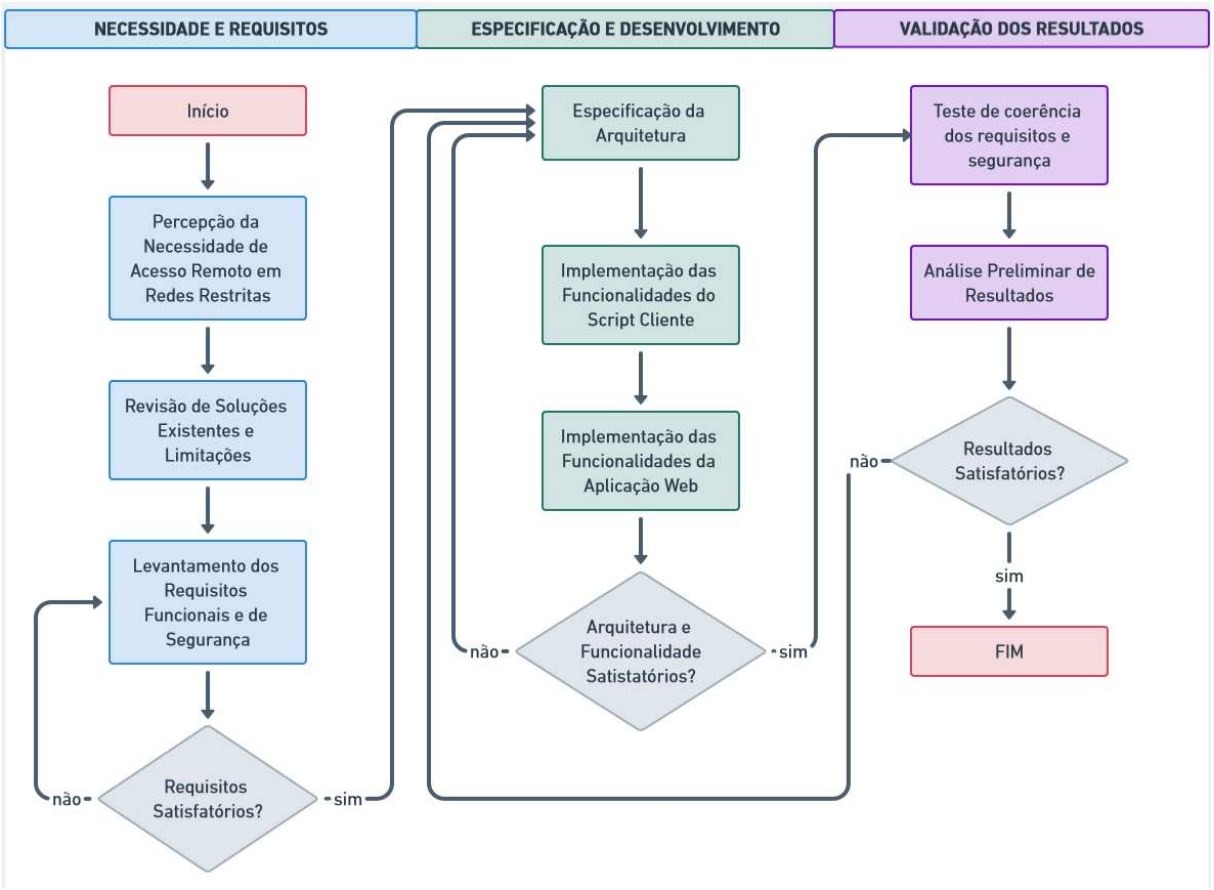

Figura 2 – Fluxograma de Atividades

3.1 Percepção da Problemática

No contexto profissional de uma empresa que desenvolve software e presta serviços de consultoria para o setor público, identificou-se a necessidade frequente de acesso re-

moto a sistemas legados instalados localmente nas máquinas dos clientes. A realização de manutenções, atualizações e suporte técnico requer uma conexão eficiente e segura com os sistemas dos clientes. No entanto, muitas organizações impõem restrições à modificação de suas infraestruturas de rede, não permitindo alterações, como a liberação de portas no *firewall* ou o redirecionamento de portas. Essa limitação dificulta o acesso remoto por parte das equipes técnicas, comprometendo a agilidade e a eficácia dos serviços prestados.

Adicionalmente, mesmo quando alguns clientes permitem ajustes mínimos, como a liberação da porta 3389 utilizada pelo RDP, surgem riscos significativos de segurança cibernética. A exposição de portas conhecidas torna os sistemas suscetíveis a ataques automatizados conduzidos por robôs (*bots*), que realizam tentativas contínuas de acesso não autorizado. Essas atividades maliciosas podem desencadear bloqueios temporários implementados pelo sistema operacional, como as medidas de proteção do Windows, resultando na indisponibilidade dos serviços e na interrupção das atividades de suporte.

Essa situação cria um cenário desafiador para a prestação de serviços de suporte técnico de forma segura, fácil e financeiramente viável. A identificação dos desafios enfrentados no acesso remoto aos sistemas dos clientes motivou a busca por uma solução que pudesse superar essas limitações.

3.2 Revisão de Soluções Existentes

O objetivo dessa etapa foi identificar soluções já propostas na literatura, analisar suas características, vantagens e limitações, e verificar como poderiam contribuir para o desenvolvimento da solução apresentada neste projeto.

A fim de compreender o panorama atual das soluções disponíveis para acesso remoto seguro em ambientes com restrições de rede, foram analisadas ferramentas e abordagens que buscam resolver problemas semelhantes. Soluções baseadas em VPN são frequentemente adotadas para estabelecer conexões seguras entre redes distintas. Embora eficazes em muitos cenários, essas soluções requerem configurações complexas e permissões para alterar a infraestrutura de rede dos clientes, o que nem sempre é possível devido a restrições administrativas ou políticas de segurança rigorosas.

Outra abordagem identificada é a utilização de túneis reversos SSH por meio de *scripts* customizados que, embora seja eficiente do ponto de vista técnico, apresenta desafios significativos em termos de usabilidade e confiabilidade da conexão. O túnel reverso precisa ser iniciado a partir da máquina cliente, o que implica que, caso o túnel seja interrompido por qualquer motivo (por exemplo, queda de conexão, reinício da máquina ou encerramento acidental do *script*), não é possível restabelecer a conexão sem acesso físico à máquina cliente. Isso representa um obstáculo considerável, especialmente quando não há pessoal técnico disponível no local para reiniciar o serviço, resultando em períodos de

indisponibilidade e impacto negativo no suporte ao cliente.

Diversas soluções *open source* exploram o uso de túneis reversos SSH para acesso remoto, contudo, carecem de mecanismos automatizados para o restabelecimento da conexão em caso de falhas, além de não oferecerem interfaces amigáveis e/ou não abordarem adequadamente questões de segurança e gerenciamento centralizado.

3.3 Levantamento de Requisitos

Para o desenvolvimento da solução proposta, foi fundamental realizar o levantamento dos requisitos essenciais, baseados no estudo das soluções da etapa anterior, que permitissem o estabelecimento de uma comunicação segura e eficiente entre o servidor e as máquinas clientes em ambientes com restrições de rede. Considerando o uso de túneis reversos SSH como núcleo da solução, tornou-se necessário compreender o contexto operacional e as necessidades específicas dos usuários.

Nesta etapa, os seguintes aspectos guiaram o levantamento de requisitos:

- **Definição dos mecanismos essenciais**: Estabelecimento dos componentes necessários para o funcionamento do sistema, incluindo o *script* em python a ser executado na máquina cliente para iniciar a conexão com o servidor, e a interface web de gerenciamento;
- **Considerações de segurança**: Identificação das medidas de segurança indispensáveis, como a utilização de autenticação por par de chaves SSH, geração de portas aleatórias no servidor, restrição de acesso ao túnel apenas ao usuário que o solicitou, e bloqueio automático das portas após o uso;
- **Automação e persistência**: Necessidade de o *script* cliente funcionar de forma automática e persistente, iniciando com o sistema operacional e restabelecendo conexões em caso de interrupções, sem requerer intervenção manual ou acesso físico à máquina cliente;
- **Usabilidade e acessibilidade**: Garantir que a solução seja fácil de usar, disponibilizando um painel de controle intuitivo para gerenciamento dos túneis e conexões, atendendo às expectativas de usuários sem expertise técnica;
- **Compatibilidade e desempenho**: Assegurar que o sistema seja compatível com diferentes sistemas operacionais nas máquinas clientes, como Windows e Linux, os quais foram escolhidos por serem os sistemas mais utilizados em ambientes corporativos e residenciais. Para novos sistemas operacionais, o *script* pode ser facilmente ajustado e testado, de modo a adaptar a configuração de permissões da chave, a obtenção do *Hardware ID* e a abertura do túnel reverso.

O objetivo deste levantamento foi estabelecer um conjunto claro de requisitos que orientassem o desenvolvimento da solução, garantindo que ela atenda às necessidades identificadas e supere as limitações das abordagens existentes.

3.4 Especificação e Desenvolvimento

Nesta etapa, materializou os requisitos funcionais definidos na etapa anterior com foco nas funcionalidades essenciais, como a autenticação segura de usuários, gerenciamento de túneis SSH reversos, e controle de acesso, visando segurança e praticidade. O desenvolvimento seguiu uma metodologia orientada por etapas, onde as tecnologias foram selecionadas com base na adequação ao projeto, como o uso de PHP com Laravel para a aplicação web e Python para o *script* cliente, garantindo integração entre os componentes. Essa etapa conclui-se com a configuração do banco de dados, estruturado para armazenar as informações essenciais de forma eficiente e segura, fundamentando o funcionamento da aplicação.

3.5 Validação e Resultados

Esta etapa de Validação e Resultados teve como objetivo verificar se o sistema desenvolvido atende aos requisitos e expectativas iniciais de forma funcional, segura e eficiente. Para isso, foram realizados diversos testes que simularam cenários reais de uso, permitindo avaliar a robustez e a confiabilidade das funcionalidades implementadas. Esses testes foram planejados para analisar aspectos fundamentais do sistema, como a comunicação entre o cliente e o servidor, a eficácia das medidas de segurança adotadas e a facilidade de uso da interface.

3.5.1 Cenário de Teste

Foram definidos alguns cenários de teste para avaliar a capacidade do mecanismo desenvolvido. Abaixo serão descritos os cenários.

- **Cenário 01 - Funcional**: Verificar o funcionamento correto das principais funcionalidades, como autenticação de usuários, gerenciamento de clientes e máquinas, abertura e fechamento de túneis SSH reversos. Aspectos analisados: **Conformidade com os Requisitos**;
- **Cenário 02 - Comunicação**: Avaliar a comunicação entre o *script* cliente e o servidor, assegurando que o *endpoint*[1] funcione conforme o esperado e que o *Hardware ID*

[1] Ponto final de comunicação de um sistema, onde ocorre a interação entre o cliente e o servidor.

seja corretamente utilizado. Aspectos analisados: **Desempenho** e **Conformidade com os Requisitos**;

- **Cenário 03 - Segurança**: Validar a segurança do sistema nos casos em que o cliente possui uma chave privada inválida ou um *Hardware ID* alterado. Aspectos analisados: **Desempenho** e **Conformidade com os Requisitos**;
- **Cenário 04 - Conexão Interrompida**: Simular a interrupção da conexão do cliente durante uma sessão ativa, para avaliar a capacidade de recuperação e gerenciamento de erros. Aspectos analisados: **Desempenho** e **Conformidade com os Requisitos**;
- **Cenário 05 - Cliente Indisponível**: Testar situações em que o cliente está temporariamente inacessível para estabelecer conexão. Avalia a resposta do sistema em condições adversas. Aspectos analisados: **Desempenho** e **Conformidade com os Requisitos**;

3.5.2 Análise de Resultados

Os resultados dos testes serão analisados para verificar se os objetivos do sistema foram atingidos. Serão considerados aspectos como:

- **Conformidade com os Requisitos**:
  - No **Cenário 01**, todas as funcionalidades previstas (autenticação, gerenciamento de clientes/máquinas e abertura/fechamento de túneis SSH reversos) foram verificadas e funcionaram conforme esperado.
  - O **Cenário 02** confirmou a correta comunicação entre o *script* cliente e o servidor, validando o uso do *Hardware ID* e o funcionamento do *endpoint*.
  - Nos **Cenários 04 e 05**, o sistema respondeu adequadamente a condições adversas, como perda de conexão ou cliente indisponível.
- **Desempenho**:
  - Nos **Cenários 02, 04 e 05**, o desempenho foi avaliado em termos de tempo de resposta e estabilidade.
  - O sistema conseguiu identificar conexões perdidas ou indisponíveis, permitindo ao usuário reagir rapidamente a esses eventos.
  - A estabilidade foi mantida nos testes de comunicação contínua entre cliente e servidor.

- **Segurança**:
  - O **Cenário 03** evidenciou que o sistema protege contra acessos não autorizados. As chaves privadas inválidas e *Hardware IDs* alterados foram corretamente rejeitados pelo servidor.
  - A integração com o *Fail2ban* também mostrou eficiência ao bloquear tentativas de força bruta após múltiplas falhas de autenticação.
  - Em todos os cenários, os dados trafegados foram protegidos pelo túnel SSH, garantindo confidencialidade e integridade.

4 Desenvolvimento do Projeto

Esta etapa detalha o processo de construção do sistema, abrangendo desde o planejamento inicial até a implementação final de cada módulo. Esse processo seguiu uma metodologia estruturada, focada em transformar os requisitos especificados em uma solução prática e funcional, garantindo o alinhamento com os objetivos do projeto.

A Figura 3 ilustra a arquitetura da solução proposta, contendo todos os componentes e interfaces previstas de comunicação.

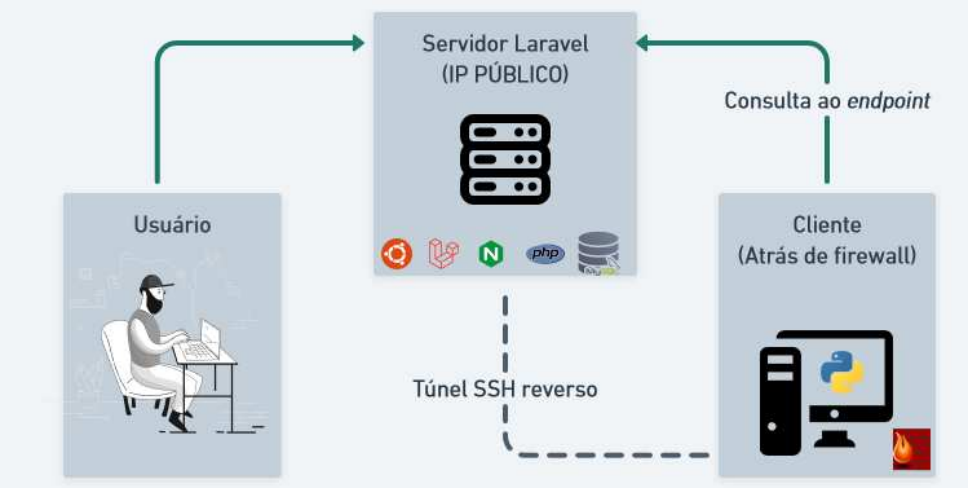

Figura 3 – Arquitetura da solução proposta.

4.1 Artefato de Requisitos

Para orientar o desenvolvimento da solução proposta e assegurar que todas as necessidades sejam atendidas, foram levantados os requisitos funcionais e requisitos não funcionais do projeto. Abaixo seguem os requisitos.

4.1.1 Requisitos Funcionais do Projeto

1. **Autenticação de Usuários**: Permitir a autenticação segura de usuários utilizando e-mail, senha e 2FA (Autenticação de Dois Fatores), garantindo que apenas usuários autorizados possam acessar a aplicação;

2. **Gerenciamento de Clientes e Máquinas**: Facilitar o cadastro, edição e exclusão de clientes e suas respectivas máquinas, associando cada máquina a um identificador único de hardware;

3. **Abertura e Fechamento de Túneis SSH Reversos**: Autorizar que usuários autenticados solicitem a abertura e fechamento de túneis SSH reversos para máquinas clientes, especificando a porta do serviço desejado;

4. **Geração de Portas Aleatórias**: Implementar a geração de portas aleatórias no servidor web dentro de um intervalo predefinido (entre 40000 e 60000) para cada túnel solicitado. Esse intervalo foi escolhido por ser pouco utilizado por serviços locais, reduzindo a probabilidade de conflitos com outras aplicações rodando no mesmo servidor;

5. **Configuração Dinâmica do *Firewall***: Automatizar a configuração do *firewall* do servidor para permitir o acesso à porta gerada apenas para o endereço IP do usuário que solicitou o túnel;

6. **Script Cliente em Python**: Incluir um *script* Python a ser executado nas máquinas clientes que consulta o *endpoint* a cada 5 segundos, para verificar novas solicitações e abrir o túnel reverso utilizando a chave privada;

7. **Endpoint para Verificação de Solicitações**: Disponibilizar um *endpoint* que permita aos *scripts* clientes verificarem periodicamente se há solicitações de conexão pendentes;

8. **Monitoramento de *Status* das Conexões**: Exibir e monitorar o *status* das conexões SSH reversas, permitindo visualizar solicitações pendentes, em andamento ou concluídas através de uma interface web intuitiva;

9. **Gerenciamento de Chaves SSH**: Armazenar a chave privada no cliente e registrar a chave pública no servidor.

4.1.2 Requisitos Não Funcionais do Projeto

1. **Tecnologia e *Framework***: Desenvolver o sistema utilizando PHP com o *framework* Laravel, seguindo o padrão de arquitetura MVC (*Model-View-Controller*) para garantir organização e manutenibilidade do código;

2. **Segurança da Comunicação**: Assegurar que toda a comunicação entre o servidor e as máquinas clientes seja realizada de forma segura, utilizando protocolos criptografados como HTTPS para proteger os dados transmitidos;

3. **Usabilidade e Interface Intuitiva**: Projetar uma interface web intuitiva e responsiva, proporcionando uma experiência de uso agradável e facilitando o gerenciamento das conexões pelos usuários mais leigos;

4. **Registro e Auditoria**: Implementar o registro de *logs* de atividades para fins de monitoramento.

Estes requisitos servem como base para o desenvolvimento da solução, garantindo que todas as funcionalidades essenciais sejam implementadas. Abaixo será detalhado as funcionalidades previstas nos componentes da arquitetura.

4.1.3 Aplicação *Script Cliente*

O objetivo principal desta aplicação é garantir que o *script* atenda aos requisitos funcionais e não funcionais do sistema, facilitando o acesso remoto seguro. O *script* cliente foi desenvolvido com funcionalidades que incluem a identificação única de cada máquina cliente, utilizando um identificador de hardware (Hardware ID) para associar solicitações de túnel a dispositivos específicos. Esse identificador é obtido por meio de métodos específicos de cada sistema operacional, como bibliotecas nativas para Windows e Linux.

A comunicação entre o *script* e o servidor web é realizada de forma periódica e segura, com uso de requisições HTTP (preferencialmente HTTPS), permitindo que o cliente verifique e responda a novas solicitações de túnel. Quando uma solicitação é detectada, o *script* inicia automaticamente uma conexão SSH reversa com o servidor, redirecionando as portas de forma compatível com diferentes plataformas. Para garantir a segurança da conexão, o *script* ajusta as permissões da chave privada utilizada, protegendo-a segundo as melhores práticas de segurança de cada sistema.

Além disso, o *script* é configurado para executar automaticamente junto com o sistema operacional e reiniciar as conexões em caso de falhas, assegurando sua persistência e reduzindo a necessidade de intervenção manual. A compatibilidade multiplataforma é garantida por meio da abstração de diferenças entre Windows e Linux, utilizando condicionais e bibliotecas apropriadas para cada ambiente. O desenvolvimento foi realizado em Python, escolhido pela sua portabilidade e pela variedade de bibliotecas disponíveis que simplificam o desenvolvimento em múltiplas plataformas.

4.1.4 Aplicação Administrador/Servidor (*Server-Broker*)

Para o desenvolvimento desta aplicação, que foi instituída como *Server-Broker*, foi utilizado a linguagem PHP com o *framework* Laravel. A aplicação foi pautada no padrão MVC, separando a lógica de negócio, a apresentação e o controle das requisições.

Os serviços oferecidos pela aplicação são:

- **Autenticação de usuários:** implementado utilizando o Laravel *Jetstream*. O *Jetstream* fornece autenticação por email e senha, além de autenticação de dois fatores,

Tabela 2 – Bibliotecas e Módulos Utilizados

Biblioteca/Módulo	**Descrição**
`Requests`	Utilizada para comunicação HTTP com o servidor web.
`Subprocess`	Executa comandos do sistema, como o estabelecimento de conexões SSH.
`Uuid` e `Platform`	Usadas para obtenção de informações do sistema e identificação de hardware.
`Wmi` (Windows)	Acessa informações de hardware específicas no Windows.
`Os` e `Sys`	Gerenciam interações com o sistema operacional e manipulação de arquivos.
`Ctypes` (Windows)	Verifica privilégios de administrador no Windows.

garantindo acesso seguro à aplicação;

- **Gerenciamento de clientes e máquinas:** permite cadastrar, editar e excluir registros;
- **Abertura e fechamento de túneis SSH reversos:** serviço que possibilita aos usuários solicitar conexões com as máquinas clientes, selecionando a porta do serviço desejado;
- **Geração de porta aleatória:** solicita a abertura de um túnel, onde o sistema gerará uma porta aleatória dentro de um intervalo predefinido (por exemplo, entre 40000 e 60000) para evitar conflitos e garantir que portas não utilizadas sejam alocadas;
- **Configuração dinâmica do *firewall*:** abre a porta gerada no *firewall* do servidor apenas para o endereço IP do usuário que acessou a aplicação;
- **Fechamento automático da porta no *firewall*:** quando o túnel for encerrado, a aplicação fechará a porta no *firewall*, garantindo que não permaneçam portas abertas desnecessariamente;
- **Monitoramento do *status* das conexões:** serviço que permite visualizar solicitações pendentes, em andamento ou concluídas. Para assegurar que a aplicação web reflita o estado atual das conexões, um *script* em PHP e verifica periodicamente o **status** dos túneis. Esse monitoramento permite identificar e atualizar automaticamente o **status** de túneis que tenham sido encerrados abruptamente devido a problemas de rede ou no lado do cliente, sem que o usuário tenha fechado manualmente pela aplicação web.

- **Implementação de um *endpoint* para comunicação com o *script cliente*:** serviço que permite as máquinas clientes verifiquem se há solicitações de conexão pendentes;
- **Uso do Hardware ID nas consultas:** serviço que assegura apenas máquinas autorizadas a estabelecer túneis.

4.1.5 Detalhes sobre o *Endpoint* e a Comunicação com o *Script Cliente*

Para permitir que o *Script Cliente* verifique se há solicitações de túnel pendentes, será implementado um *Endpoint* específico na aplicação *server-broker*. O *Endpoint* está acessível através de uma URL definida, por exemplo:

```
https://ip-servidor.com/api/check-tunnel-request
```

O *script* cliente realiza requisições periódicas para este *Endpoint*, enviando no corpo da requisição o *Hardware ID* da máquina cliente. O *Hardware ID* é utilizado para identificar unicamente cada máquina e associá-la às suas respectivas solicitações de conexão.

Exemplo de corpo da requisição JSON que o *script* cliente envia:

```
{
    "hardware_id": "ID_UNICO_DA_MAQUINA"
}
```

Ao receber a requisição, o servidor realiza os seguintes passos:

- Verifica se existe uma máquina cadastrada com o *Hardware ID* fornecido;
- Caso exista, busca por solicitações de conexão com *status pending* associadas a esta máquina;
- Se encontra uma solicitação pendente, retorna uma resposta contendo as informações necessárias para o estabelecimento do túnel.

Exemplo de resposta JSON que o servidor envia quando houver uma solicitação pendente:

```
{
    "success": true,
    "service_port": 3389,
```

```
    "server_port": 45000,
    "status": "pending"
}
```

Nesta resposta:

- `success`: indica o resultado da operação. `true` significa que a operação foi bem-sucedida e há uma solicitação pendente. `false` indica que não há conexões a serem estabelecidas;
- `service_port`: a porta do serviço na máquina cliente que deve ser acessada (por exemplo, porta do RDP);
- `server_port`: a porta no servidor que foi alocada para o túnel SSH reverso.
- `status`: o *status* atual da solicitação (`pending`, `in_progress`, etc.).

Caso não haja solicitações pendentes, o servidor retorna uma resposta indicando que não há conexões a serem estabelecidas:

```
{
    "success": false,
    "message": "Nenhuma solicitação de túnel pendente."
}
```

4.1.6 Uso do *Hardware ID* nas Consultas

O *Hardware ID* é um identificador único obtido a partir das características de hardware da máquina cliente. Sua utilização é fundamental para:

- Garantir que apenas máquinas previamente cadastradas e autorizadas possam estabelecer túneis com o servidor;
- Associar as solicitações de conexão à máquina correta, evitando conflitos e reforçando a segurança.

Na aplicação administrativa, ao cadastrar uma nova máquina, o administrador informará o *Hardware ID* correspondente. Dessa forma, quando o *script cliente* enviar o *Hardware ID* no *endpoint*, o servidor conseguirá identificar a máquina e verificar se há solicitações pendentes para ela.

4.1.7 Interface de Comunicação entre o *Script Cliente* e o *Server-Broker*

A Tabela 3 ilustra o formato de comunicação entre o *Script Cliente* e o *Server-Broker*.

Tabela 3 – Fluxo de Comunicação entre o *Script Cliente* e o Servidor

Etapa	**Descrição**
`Requisição do Script Cliente`	O *Script Cliente*, executando na máquina cliente, obterá periodicamente o *Hardware ID* e realizará uma requisição POST para o *endpoint* `/api/check-tunnel-request`, enviando o *Hardware ID* no corpo da requisição.
`Verificação no Servidor`	O servidor receberá a requisição, verificará se há uma máquina cadastrada com aquele *Hardware ID* e se existem solicitações de conexão pendentes.
`Resposta do Servidor`	Se houver uma solicitação pendente, o servidor responderá com as informações necessárias para o *script* estabelecer o túnel SSH reverso (portas, *status*).
`Estabelecimento do Túnel`	O *Script Cliente* utilizará essas informações para iniciar o túnel SSH reverso, conectando-se ao servidor na porta especificada.
`Atualização de Status`	O servidor atualizará o *status* da solicitação de conexão para `in_progress` ou `completed`, conforme o caso.

Este mecanismo permite uma comunicação segura e eficiente entre o *Script Cliente* e o servidor, facilitando o estabelecimento dos túneis SSH reversos somente quando necessário e apenas para máquinas autorizadas.

Para a execução adequada do *Script Cliente*, foram realizadas adaptações específicas para ambientes Windows e Linux, buscando garantir a compatibilidade e a segurança da conexão SSH reversa. A seguir, descrevem-se os principais aspectos de sua implementação:

4.1.7.1 Ajuste de Permissões da Chave Privada

Para assegurar a proteção da chave privada SSH em sistemas Windows, o *script* remove a herança de permissões do arquivo e ajusta o acesso para que apenas o usuário atual tenha permissão de leitura. Essa configuração reduz a possibilidade de acesso indevido à chave privada por outros usuários no sistema.

4.1.7.2 Validação da Chave SSH

Antes de iniciar o túnel SSH, o *script* verifica a validade da chave SSH, assegurando que ela seja aceita pelo servidor. Essa verificação prévia previne tentativas de conexão desnecessárias e reforça a segurança ao garantir que a chave configurada é válida para autenticação no servidor.

4.1.7.3 Verificação de Privilégios no Windows

Em sistemas Windows, o *script* confirma se está sendo executado com privilégios de administrador, condição necessária para ajustar as permissões da chave privada. Esse procedimento evita erros na configuração e garante que a chave privada esteja adequadamente protegida.

4.1.7.4 Identificador de Hardware (*Hardware ID*)

O identificador de hardware é obtido de maneira específica para cada sistema operacional. Em sistemas Windows, o *script* utiliza informações como `ProcessorId` e `SerialNumber`, enquanto em sistemas Linux utiliza o endereço MAC da interface de rede, obtido via `uuid.getnode`. Esse identificador é utilizado para autenticação segura, assegurando que apenas máquinas autorizadas estabeleçam conexões com o servidor.

4.1.7.5 Comando SSH para Estabelecimento de Túnel

O *script* utiliza o comando SSH com parâmetros específicos para estabelecer o túnel reverso:

```
ssh -i chave_ssh -o BatchMode=yes -o StrictHostKeyChecking=no -R
{server_port}:localhost:{service_port} {USERNAME_SSH}@{SERVER_ADDRESS} -N
```

Esse comando configura o túnel SSH reverso de modo seguro e automatizado, sem requerer interação do usuário.

4.1.7.6 Execução em *Thread* Separada

Para permitir que o *script* continue consultando o servidor enquanto o túnel está ativo, o túnel SSH é executado em uma *thread* separada. Essa abordagem permite o estabelecimento de múltiplas conexões simultâneas, assegurando que o *script* possa abrir mais de um túnel quando necessário.

4.1.8 Banco de Dados

O sistema utiliza o banco de dados MySQL versão 8.0.39 para armazenar informações essenciais ao seu funcionamento. As principais tabelas e seus relacionamentos são apresentados no diagrama ER (Entidade-Relacionamento) ilustrado na Figura 4.

As tabelas que compõem o banco de dados são:

- **users**: Armazena os dados dos usuários do sistema, incluindo informações de autenticação, como nome, e-mail, senha e detalhes relacionados à 2FA. Possui uma relação um-para-muitos com a tabela *clients*;

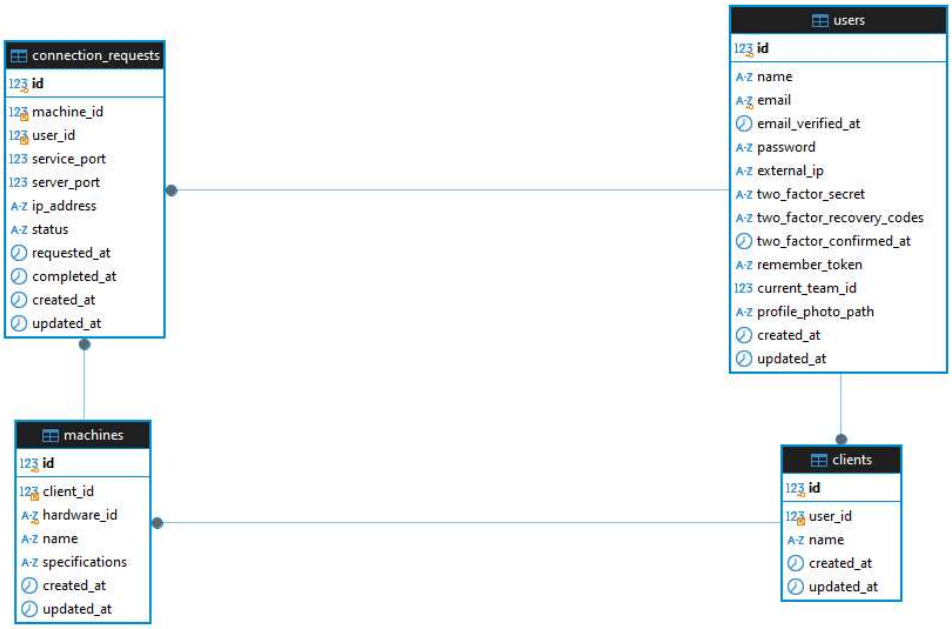

Figura 4 – Modelo ER do Banco de Dados.

- **clients**: Representa os clientes atendidos pelo sistema. Cada cliente está associado a um usuário que é responsável por seu gerenciamento. A relação é estabelecida através da chave estrangeira `user_id`;
- **machines**: Contém informações sobre as máquinas dos clientes que podem estabelecer túneis reversos SSH. Cada máquina está associada a um cliente por meio da chave estrangeira `client_id`. Possui atributos como `hardware_id` (identificador único da máquina), nome e especificações;
- **connection_requests**: Registra as solicitações de conexão para túneis reversos SSH. Inclui detalhes como a porta do serviço na máquina cliente (`service_port`), a porta alocada no servidor (`server_port`), o endereço IP do solicitante (`ip_address`) e o *status* da solicitação (`status`). Está relacionada às tabelas *machines* e *users* através das chaves estrangeiras `machine_id` e `user_id`, respectivamente. Além disso, serve também como *log*, permitindo o monitoramento histórico de todas as conexões realizadas, incluindo tempo de duração e configurações utilizadas.

As relações entre as tabelas são fundamentais para o funcionamento da aplicação:

- Um **usuário** pode gerenciar vários **clientes**;
- Um **cliente** possui várias **máquinas**;
- Uma **máquina** pode ter várias **solicitações de conexão**;
- Um **usuário** pode realizar várias **solicitações de conexão**

4.1.9 Compartilhamento de Chave (Negociação dos Parâmetros de Acesso)

A metodologia para o compartilhamento de chaves e negociação dos parâmetros de acesso foi focada em garantir uma comunicação segura entre o servidor e as máquinas clientes, possibilitando o estabelecimento confiável dos túneis SSH reversos. O objetivo foi definir processos que permitam autenticação segura, evitando interceptações ou acessos não autorizados.

Para isso, foram consideradas as seguintes abordagens:

- **Geração de chaves SSH no servidor**: As chaves SSH foram geradas no servidor web (Ubuntu) para cada cliente, utilizando o algoritmo RSA de 4096 bits com um hash SHA-256. A geração das chaves foi realizada com o comando apropriado do OpenSSH, resultando em um par de chaves (pública e privada) nomeadas, por exemplo, como `chave_ssh`;
- **Distribuição da chave privada ao cliente**: A chave privada gerada no servidor foi transferida manualmente para a máquina cliente que executou o *Script Cliente*. Para garantir a segurança da chave durante a transferência, foi utilizado um canal seguro, como uma conexão SSH pré-estabelecida. Além disso, foi adotado o cuidado de realizar a transferência diretamente para o cliente autorizado, minimizando o risco de interceptações;
- **Configuração do cliente para uso da chave privada**: O *Script Cliente* no cliente foi configurado para utilizar a chave privada (`chave_ssh`) ao estabelecer a conexão SSH com o servidor. Foram ajustadas as permissões do arquivo de chave privada conforme necessário para o correto funcionamento e segurança;
- **Registro da chave pública no servidor**: A chave pública correspondente foi adicionada ao arquivo `authorized_keys` do servidor SSH, permitindo que a máquina cliente se autentique utilizando a chave privada correspondente;
- **Autenticação baseada em chave pública**: Com a chave pública registrada no servidor e a chave privada configurada no cliente, a autenticação das conexões SSH foi realizada por meio de chaves, eliminando a necessidade de senhas e aumentando a segurança das conexões;
- **Negociação dos parâmetros de acesso**: Os parâmetros necessários para o estabelecimento dos túneis (como as portas a serem utilizadas) foram fornecidos ao cliente através do *endpoint* implementado na aplicação *Server Broker*, conforme descrito anteriormente. O uso do *Hardware ID* garantiu que apenas máquinas autorizadas recebessem essas informações;

- **Segurança na transmissão de dados**: Toda a comunicação entre as máquinas clientes e o servidor, especialmente durante a negociação dos parâmetros de acesso, foi realizada de forma segura, utilizando protocolos criptografados (como HTTPS).

Essa abordagem permitiu um controle centralizado das chaves SSH, facilitando a gestão e revogação de acesso quando necessário. Ao gerar as chaves no servidor e distribuir a chave privada ao cliente, assegura-se que cada cliente possui uma chave única associada, reforçando a segurança das conexões.

4.1.10 Configuração do Servidor de Aplicação

A aplicação administrador/servidor (*Server-Broker*) está hospedada em um VPS *(Virtual Private Server)* fornecido pela empresa Hostinger, localizada em São Paulo, Brasil. A seguir, apresentamos as especificações técnicas do servidor utilizado:

Tabela 4 – Especificações da VPS Utilizada

Item	**Especificação**
Empresa	Hostinger
Localização	São Paulo, Brasil
Sistema Operacional	Ubuntu 24.04.1 LTS
Memória	8 GB RAM
Processador	AMD EPYC 9354P (2 VCPU)
Virtualização	AMD-V (KVM)

A Tabela 5 ilustra as principais ferramentas instaladas no servidor para comportar todos os serviços descritos.

Tabela 5 – Versões dos Softwares Utilizados

Software	**Versão**
Laravel PHP Framework	11.24.1
PHP: Hypertext Preprocessor	8.3.6
Zend Engine	v4.3.6
MySQL Database	8.0.39
Nginx Web Server	1.24.0
Python Programming Language	3.12.6

Abaixo será apresentado as ações realizadas para configuração do servidor para comportar a aplicação administrador/servidor (*Server Broker*). Dentre as ações planejadas incluem:

- **Escolha do Sistema Operacional**: Ubuntu Server como sistema operacional do servidor, devido à sua estabilidade, segurança e amplo suporte da comunidade;
- **Instalação dos Serviços Necessários**: Serviços essenciais como servidor web Nginx, PHP, MySQL e OpenSSH para suportar a aplicação Laravel e as conexões SSH;
- **Configuração do Servidor SSH**: Configuração do serviço SSH para permitir túneis reversos, ajustando as diretivas necessárias no arquivo `sshd_config`, como `GatewayPorts yes`;
- **Configuração do *Firewall***: O *firewall UFW (Uncomplicated Firewall)* foi utilizado para gerenciar as portas de entrada e saída, garantindo que apenas as portas necessárias estejam abertas e que o acesso seja restrito aos IPs autorizados;
- **Implementação de Medidas de Segurança**: Instalação do *Fail2Ban* para proteção contra tentativas de acesso não autorizadas, atualização regular do sistema e aplicação de políticas de segurança recomendadas;
- **Configuração de HTTPS**: Configuração dos certificados SSL/TLS para habilitar o protocolo na aplicação web, assegurando a criptografia das comunicações entre os usuários e o servidor.

4.1.11 Fluxo de Funcionamento do Sistema

O funcionamento do sistema segue o fluxo descrito a seguir:

1. **Solicitação de Abertura de Túnel**: O usuário autenticado acessa o painel de controle e solicita a abertura de um túnel para uma máquina específica, informando a porta do serviço desejado (por exemplo, porta 3389 para RDP);
2. **Geração de Porta Aleatória**: A aplicação web gera uma porta aleatória no servidor dentro de um intervalo pré-definido (entre 40000 e 60000) e verifica se a porta está disponível, evitando conflitos;
3. **Registro da Solicitação**: A solicitação de conexão é registrada no banco de dados com *status pending*, incluindo informações como a porta do serviço, a porta do servidor e o endereço IP do usuário;
4. **Configuração do *Firewall***: O servidor abre a porta gerada no *firewall*, permitindo acesso apenas a partir do endereço IP externo do usuário que solicitou o túnel, aumentando a segurança;

5. **Verificação pelo *Script* Cliente**: O *script* cliente, em execução na máquina remota, verifica periodicamente (a cada 5 segundos) se há solicitações de túnel pendentes para o seu *hardware_id* através de um *endpoint* fornecido pela aplicação web;

6. **Estabelecimento do Túnel Reverso**: Ao identificar uma solicitação pendente, o *script* cliente estabelece o túnel reverso SSH utilizando o comando apropriado, redirecionando a porta local do serviço para a porta remota gerada no servidor;

7. **Atualização do *Status***: A aplicação web detecta que o túnel foi estabelecido (através da verificação da porta no servidor) e atualiza o *status* da solicitação para *in_progress*;

8. **Acesso ao Serviço Remoto**: O usuário pode então acessar o serviço remoto através do endereço do servidor e da porta gerada, com acesso restrito ao seu endereço IP;

9. **Fechamento do Túnel**: Quando o acesso não é mais necessário, o usuário pode fechar o túnel através da interface web. O servidor encerra o processo SSH correspondente, fecha a porta no *firewall* e atualiza o *status* da solicitação para *completed*.

4.2 Implementação da Aplicação Web

A aplicação web desenvolvida em Laravel é responsável por gerenciar clientes, máquinas e solicitações de túnel. A seguir, são destacadas as funcionalidades principais e as soluções implementadas.

4.2.1 Geração de Portas Aleatórias e Verificações

Ao solicitar a abertura de um túnel, a aplicação precisa gerar uma porta aleatória no servidor para estabelecer o túnel reverso. Para isso, foi implementada uma função que gera uma porta aleatória dentro de um intervalo seguro e verifica se a porta está disponível, tanto no sistema operacional quanto no banco de dados (para evitar conflitos com outras solicitações em andamento).

```
public function generateRandomAvailablePort() {
    $minPort = 40000;
    $maxPort = 60000;
    do {
        $port = rand($minPort, $maxPort);
    } while ($this->isPortInUse($port));
    return $port;
}
```

```
private function isPortInUse($port) {
    // Verifica no sistema operacional
    $output = shell_exec("lsof -i :$port");
    if (!empty($output)) {
        return true;
    }
    // Verifica no banco de dados
    return ConnectionRequest::where('server_port', $port)
        ->whereIn('status', ['pending', 'in_progress'])
        ->exists();
}
```

4.2.2 Abertura de Túnel e Registro da Solicitação

Ao receber a solicitação do usuário, a aplicação realiza as seguintes ações:

- Valida os dados de entrada, assegurando que a porta do serviço no cliente esteja dentro do intervalo de 1 a 65535, que corresponde ao total de portas TCP/UDP disponíveis, garantindo suporte a qualquer serviço que o usuário deseje acessar no *localhost*;
- Gera uma porta aleatória disponível no servidor dentro do mesmo intervalo (1 a 65535), uma vez que o sistema deve acomodar qualquer configuração de serviços locais que possam estar rodando no servidor;
- Adiciona uma regra no *firewall* do servidor para permitir acesso à porta gerada apenas a partir do endereço IP externo do usuário que fez a solicitação;
- Registra a solicitação no banco de dados com *status pending*.

4.2.3 Atualização do *Status* da Solicitação

A aplicação monitora o *status* do túnel verificando se a porta gerada está em uso (indicando que o túnel foi estabelecido pelo *script* cliente). Ao detectar que a porta está em uso, o *status* da solicitação é atualizado para *in_progress*.

Para assegurar o monitoramento contínuo dos túneis SSH reversos, foi implementado um *script* em PHP que é executado a cada 2 segundos. Este *script* verifica se as portas associadas aos túneis estão abertas no servidor, utilizando o comando `nc` (netcat). Se o *script* detectar que uma porta previamente aberta não está mais em uso (indicando que o túnel foi encerrado inesperadamente devido a problemas de rede ou no cliente),

ele atualiza o *status* do túnel no banco de dados e aciona os procedimentos necessários para encerrar o túnel adequadamente no sistema. Dessa forma, a aplicação web permanece sincronizada com o estado real das conexões, permitindo um gerenciamento eficaz. A execução periódica do *script* é gerenciada pelo *Supervisor*[1] do Linux, que garante sua execução contínua e reinicialização automática em caso de falhas.

4.3 Considerações de Segurança

Diversas medidas de segurança foram implementadas para garantir a confiabilidade do sistema:

- **Autenticação Segura no Servidor Web**: O acesso à aplicação web é protegido por um sistema de autenticação que requer login com e-mail e senha. Além disso, foi implementada a opção de autenticação 2FA. Isso adiciona uma camada extra de segurança, dificultando o acesso não autorizado mesmo em caso de comprometimento da senha;
- **Comunicação Segura**: É recomendado o uso de HTTPS para a comunicação entre o *script* cliente e o *endpoint* da aplicação web, protegendo os dados transmitidos;
- **Autenticação por Chave Privada**: O *script* cliente utiliza uma chave privada para autenticação no servidor SSH, evitando o uso de senhas e aumentando a segurança;
- **Restrições de Acesso no *Firewall***: As portas abertas no servidor são configuradas para aceitar conexões apenas do endereço IP externo do usuário que solicitou o túnel, reduzindo a superfície de ataque;
- **Geração de Portas Dinâmicas**: As portas são geradas aleatoriamente e utilizadas temporariamente, dificultando ataques direcionados;
- **Validações de Entrada**: Todas as entradas de usuário são validadas na aplicação web para prevenir vulnerabilidades como injeção de comandos.

[1] Programa que permite controlar e monitorar processos no sistema operacional Linux, garantindo que serviços críticos estejam sempre em execução.

5 Testes e Experimentos

Este capítulo apresenta os cenários testados para avaliar o funcionamento do sistema proposto, destacando as diferentes condições em que o túnel SSH reverso pode ser estabelecido ou falhar.

5.1 Cenário 1 - Ideal (cliente e servidor possuem todas as configurações)

Neste cenário, tanto o cliente quanto o servidor possuem todas as configurações e chaves necessárias para o estabelecimento do túnel SSH reverso, representando o ambiente ideal de funcionamento.

O *script* cliente inicia sua execução obtendo o *Hardware ID* e localizando o arquivo da chave privada. Após ajustar as permissões da chave para que esta possa ser utilizada, verifica sua validade com o servidor. Esse ajuste garante que o sistema de segurança do sistema operacional permita o uso correto da chave. Em seguida, o script realiza uma requisição ao servidor, passando o *Hardware ID* para verificar se há uma solicitação de conexão pendente. Neste caso, o servidor responde positivamente, indicando as portas de serviço e do túnel. O *script*, então, inicia o processo de conexão SSH, estabelecendo o túnel de maneira bem-sucedida.

Após a conexão ser estabelecida, o *script* continua verificando periodicamente o *endpoint* do servidor para caso tenha mais algum pedido de conexão. Caso não haja solicitações pendentes, o *script* retorna ao seu ciclo de verificação a cada 5 segundos. Esse processo é ilustrado na Figura 5.

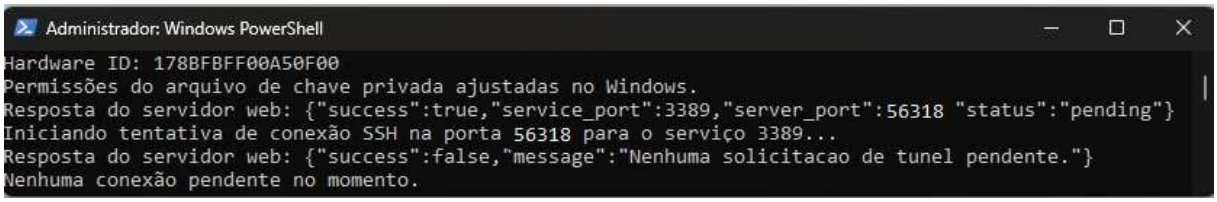

Figura 5 – Cenário ideal com conexão e túnel SSH reverso estabelecidos

Para que esses resultados fossem alcançados, o usuário acessou o painel de controle, selecionou o cliente desejado, digitou a porta do serviço (por exemplo, 3389) e clicou em `Abrir Túnel`. Enquanto o *script* do cliente realiza a verificação no *endpoint*, a mensagem `Conectando...` é exibida para o usuário. Assim que a conexão é bem-sucedida, a men-

sagem muda para `Fechar Túnel`, e a porta aleatória gerada é liberada para o IP externo do usuário.

Nas Figuras 6 e 7, são apresentadas as diferenças nas tentativas de acesso realizadas com um IP autorizado e com um IP externo não autorizado. Na Figura 6, a porta está aberta, permitindo que o serviço do RDP seja reconhecido. Em contraste, a Figura 7 ilustra que a tentativa de acesso a partir de um IP não autorizado resulta em uma porta não aberta, indicando que o serviço não está acessível.

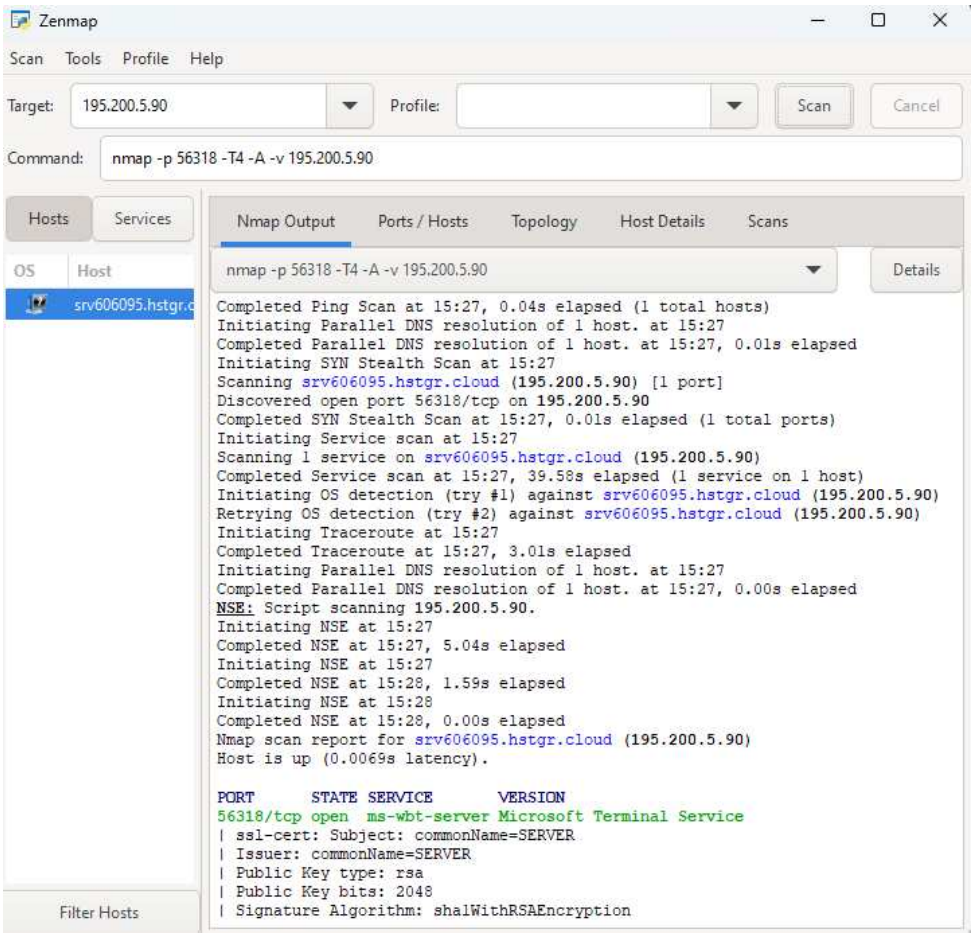

Figura 6 – Resultado do Nmap: Tentativas de acesso com IP autorizado.

Ao tentar acessar o RDP a partir de um IP autorizado, a janela de autenticação (usuário e senha) é exibida, permitindo que a conexão seja estabelecida normalmente, como ilustrado na Figura 8. Por outro lado, ao tentar o acesso a partir de um IP não autorizado, a janela de autenticação não aparece, e uma mensagem de erro indica que o serviço não está acessível (Figura 9), reforçando a segurança do sistema ao limitar o acesso apenas a endereços IP previamente autorizados.

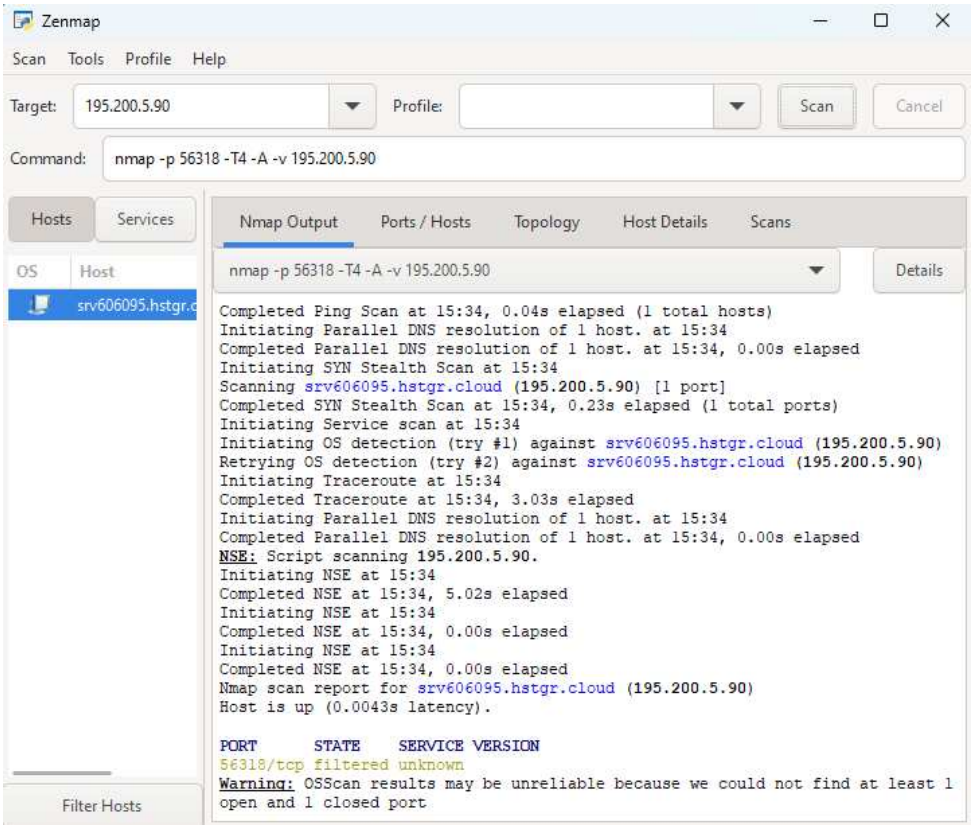

Figura 7 – Resultado do Nmap: Tentativas de acesso com IP não autorizado.

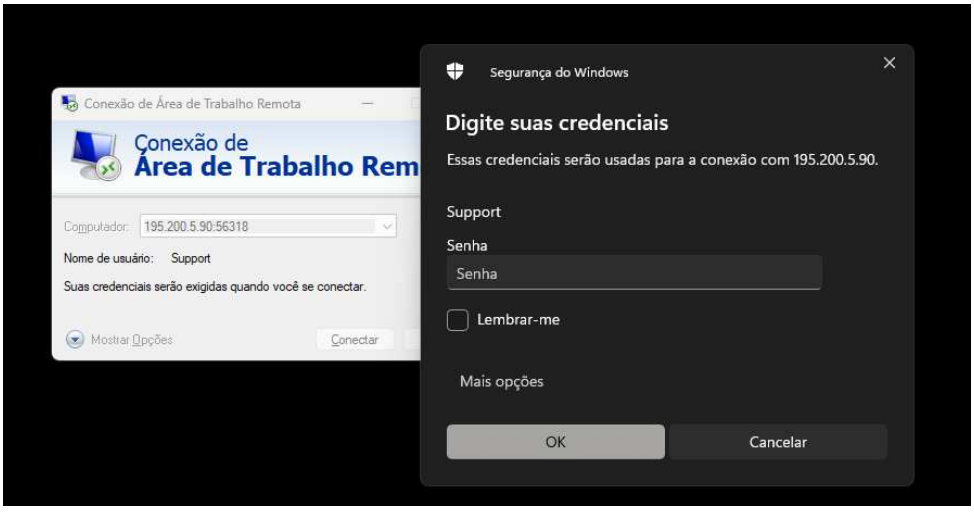

Figura 8 – Tentativa de acesso RDP com IP autorizado.

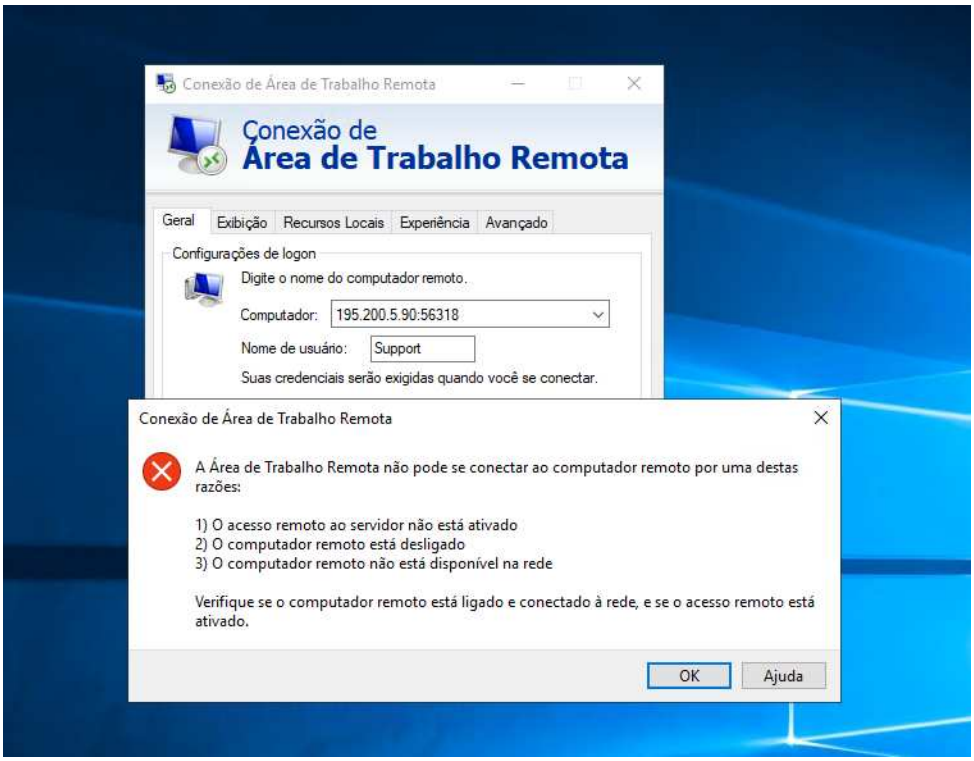

Figura 9 – Tentativa de acesso RDP com IP não autorizado.

Os *logs* do servidor SSH no servidor web ilustram esse processo. Na primeira mensagem do *log*, vemos a porta aleatória sendo liberada exclusivamente para o IP externo do solicitante. Em seguida, o comando lsof é utilizado para verificar que a conexão permanece ativa na porta designada. Finalmente, quando o usuário clica em `Fechar Túnel`, a regra de *firewall* é removida, fechando a porta para todos os IPs e encerrando a conexão.

```
root@srv606095:/var/www/laravel_project/ssh-tunel# cat /var/log/auth.log | grep 56318
2024-11-12T18:07:32.974057+00:00 srv606095 sudo: www-data : PWD=/var/www/laravel_project/ssh-tunel/public
 ; USER=root ; COMMAND=/usr/sbin/ufw allow from 186.210.235.183 to any port 56318
2024-11-12T18:33:34.344737+00:00 srv606095 sudo: www-data : PWD=/var/www/laravel_project/ssh-tunel/public
 ; USER=root ; COMMAND=/usr/bin/lsof -t -i :56318
2024-11-12T18:33:34.420390+00:00 srv606095 sudo: www-data : PWD=/var/www/laravel_project/ssh-tunel/public
 ; USER=root ; COMMAND=/usr/sbin/ufw delete allow from 186.210.235.183 to any port 56318
root@srv606095:/var/www/laravel_project/ssh-tunel#
```

Figura 10 – Log do SSH do Servidor Web.

5.2 Cenário 2 - Cliente possui uma chave SSH inválida.

Neste cenário, o cliente possui uma chave privada, mas ela é inválida para estabelecer o túnel SSH reverso.

Ao iniciar, o *script* obtém o *Hardware ID*, verifica a existência do arquivo de chave privada e ajusta suas permissões. Quando a chave é inválida, o *script* tenta autenticar com o servidor e recebe um código de erro, indicando que a chave SSH foi rejeitada. Como resposta, o *script* exibe uma mensagem de falha e encerra sua execução, sem prosseguir com novas tentativas de verificação, conforme mostrado na Figura 11.

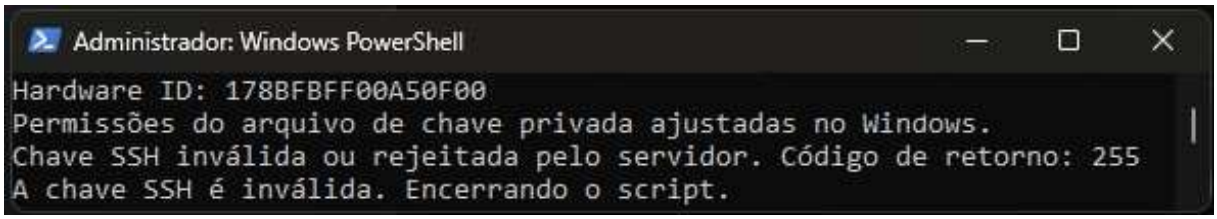

Figura 11 – Mensagem de erro ao tentar autenticar com uma chave privada inválida

No lado do servidor, o *log* SSH registra a tentativa de autenticação e exibe uma mensagem de falha ao identificar que a chave privada utilizada pelo cliente é inválida. Esse log, mostrado na Figura 12, documenta a rejeição da chave e permite o monitoramento de tentativas de acesso não autorizadas.

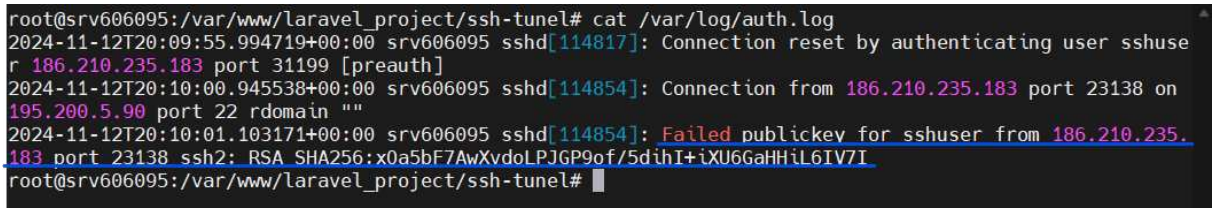

Figura 12 – Log do servidor SSH indicando falha de autenticação devido a chave privada inválida.

5.3 Cenário 3 - Cliente possui a chave, mas o *Hardware ID* mudou

Neste cenário, o cliente possui a chave privada necessária para autenticação, mas o identificador de hardware (*Hardware ID*) diverge do esperado pelo servidor, simulando uma alteração no dispositivo do cliente.

O *script* cliente localiza a chave privada, ajusta suas permissões e valida sua autenticidade por meio de uma conexão SSH inicial com o servidor. Após essa etapa, o *script* obtém o *Hardware ID* atual e consulta o *endpoint*. Como o *Hardware ID* não está registrado, o servidor retorna um erro indicando que a máquina não foi encontrada.

Após a tentativa inicial, o *script* consulta o *endpoint* a cada 5 segundos, mas sem realizar novamente a validação da chave privada.

Embora o servidor permita sucessivas consultas ao *endpoint*, as tentativas de autenticação SSH são protegidas contra ataques de força bruta pelo *Fail2ban*[1], que bloqueia IPs após quatro falhas consecutivas.

A Figura 13 exibe a mensagem de erro gerada pelo *script* quando o *Hardware ID* não é encontrado no servidor.

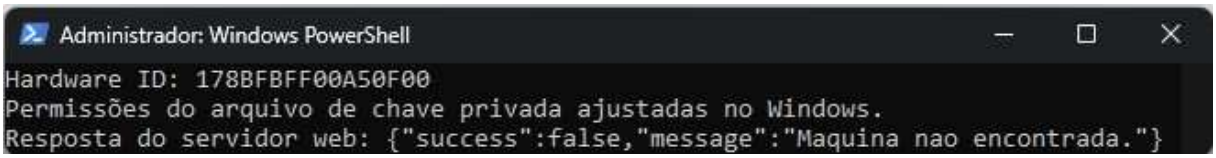

Figura 13 – Mensagem de erro do *script* ao não encontrar o *Hardware ID*.

A Figura 14 mostra o *log* do servidor indicando a ausência do *Hardware ID* registrado para a conexão.

[1] Software de prevenção de intrusão que protege servidores contra ataques de força bruta, monitorando logs e banindo endereços IP suspeitos.

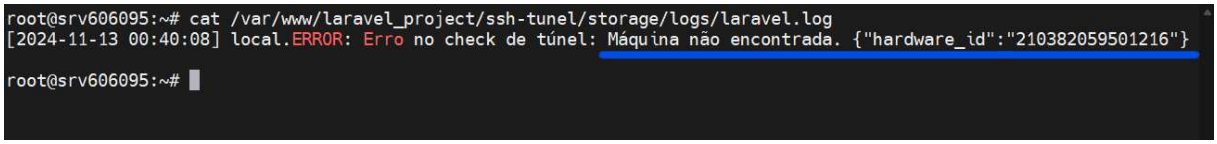

Figura 14 – Log do servidor ao não encontrar o *Hardware ID*.

5.4 Cenário 4 - Conexão do cliente é interrompida no meio da sessão

Neste cenário, a conexão do cliente com o servidor é interrompida durante uma sessão ativa, simulando problemas de conectividade.

Quando a conexão é perdida, o servidor web exibe a mensagem `Conexão Perdida` no painel de gerenciamento, no lugar da opção `Fechar Túnel`. O usuário pode então passar o mouse sobre a mensagem e visualizar a opção `Abrir Túnel`, permitindo uma nova tentativa de reconexão.

Na Figura 15, observa-se a mensagem exibida no painel ao detectar a interrupção na conexão com o cliente.

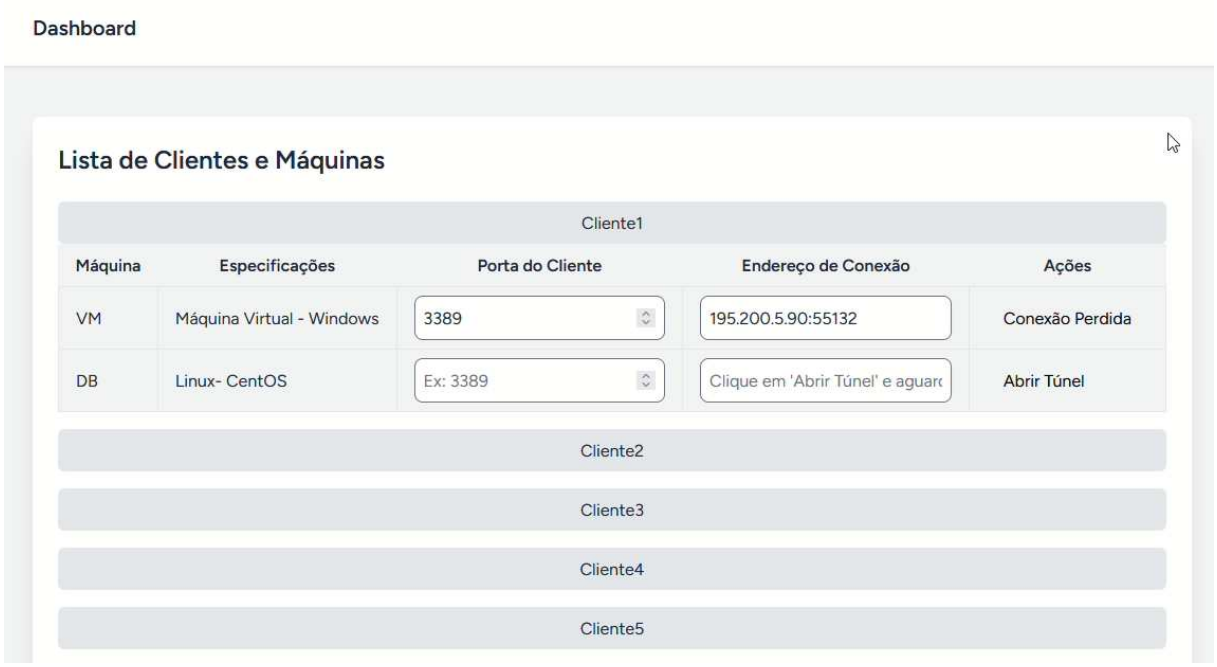

Figura 15 – Mensagem do servidor web indicando conexão perdida.

5.5 Cenário 5 - Cliente está temporariamente indisponível ou inacessível

Neste cenário, o cliente está indisponível ou inacessível no momento em que o servidor tenta estabelecer a conexão.

Quando o usuário clica em `Abrir Túnel`, o painel exibe a mensagem `Conectando...`, indicando que está aguardando a resposta do cliente. Se o cliente não responde em até 7 segundos, o servidor altera o *status* para `Sem Conexão` e exibe a opção `Cancelar Túnel` ao passar o mouse sobre a mensagem. Essa funcionalidade permite que o usuário cancele a tentativa de conexão, evitando espera desnecessária.

Na Figura 16, vemos a mensagem exibida para o usuário após a falha de conexão.

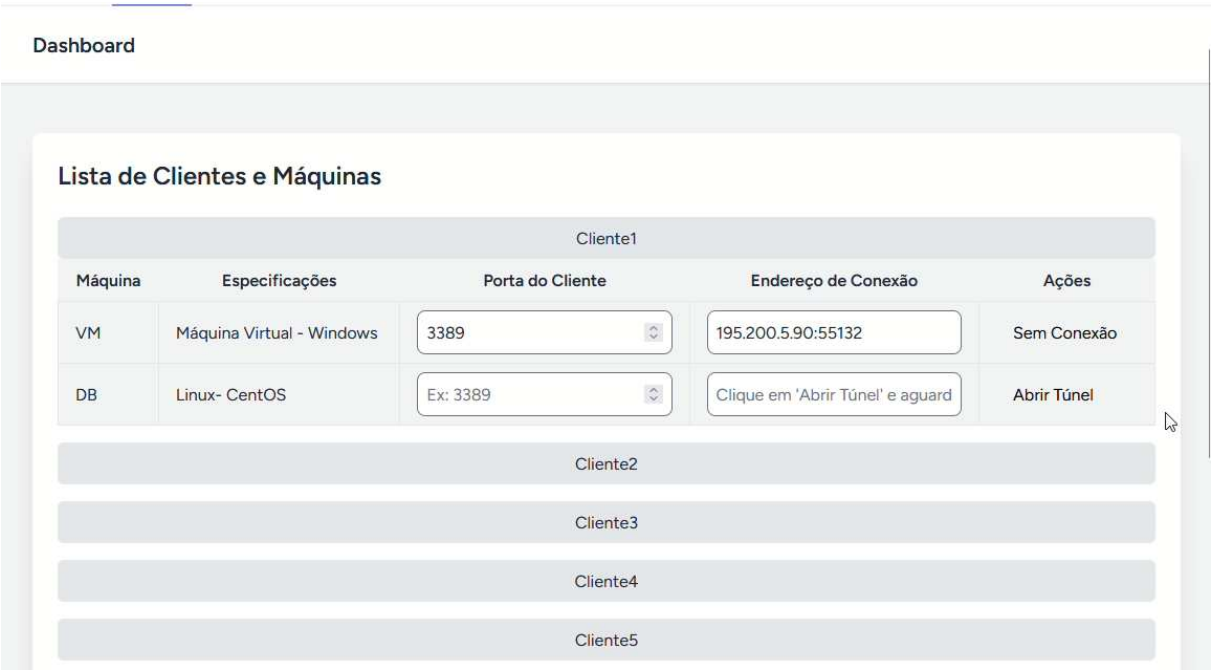

Figura 16 – Mensagem exibida para o usuário indicando que o cliente está indisponível.

6 Análise de Resultados

O sistema foi testado em ambientes controlados e reais, demonstrando eficácia na abertura e fechamento de túneis reversos SSH sem a necessidade de alterar configurações de *firewall* nos clientes. Os principais aspectos observados foram:

- **Facilidade de Uso**: A interface web permite que usuários sem conhecimentos avançados abram e fechem túneis de forma intuitiva;
- **Segurança Aprimorada**: As medidas implementadas garantem um alto nível de segurança, evitando exposição desnecessária de portas e serviços;
- **Flexibilidade**: O sistema suporta qualquer serviço, desde que esteja acessível via `localhost` e em execução na máquina cadastrada. Isso permite sua adaptação a diversas necessidades, como acesso remoto via RDP, conexões a bancos de dados e servidores web;
- **Isolamento de Usuários e Chaves**: Configurar cada cliente com um usuário exclusivo no servidor Linux, associado a uma chave SSH própria, reforça a segurança. O OpenSSH rejeita tentativas de acesso feitas com chaves de outros usuários, garantindo isolamento e prevenindo acessos não autorizados;
- **Considerações sobre Latência**: Embora essa abordagem introduza um servidor intermediário na comunicação, o que pode resultar em um aumento de latência em comparação com o redirecionamento direto de portas no *firewall* do cliente, os testes realizados demonstraram que essa latência adicional é geralmente aceitável para a maioria dos casos. No entanto, a localização do servidor intermediário é um fator importante. Caso a máquina cliente precise transferir grandes volumes de dados, o servidor deve estar mais próximo do cliente. Por outro lado, se a interação do usuário com o sistema for sensível à latência (como no RDP), o servidor deve estar mais próximo do usuário. Já uma localização central pode oferecer um bom equilíbrio em cenários híbridos.

Por meio dos testes, foi observado que a latência introduzida pelo servidor intermediário não comprometeu significativamente a experiência do usuário em aplicações como acesso remoto via RDP, conexões SSH, ou do *Cockpit* do Linux (porta 9090). Dessa forma, o sistema se mostra eficiente e seguro para o acesso remoto, com a ressalva de que a infraestrutura de rede deve ser planejada considerando as características da comunicação entre cliente e usuário para otimizar a latência das conexões.

7 Considerações Finais

O desenvolvimento deste projeto demonstrou a viabilidade de utilizar túneis reversos SSH para prover acesso remoto seguro em ambientes corporativos com restrições de rede. A integração entre a aplicação web em Laravel e o *script* cliente em Python permitiu a implementação de uma solução eficiente, alinhada a boas práticas de segurança.

O sistema atende aos objetivos propostos, oferecendo uma alternativa viável para equipes técnicas que necessitam de acesso remoto sem interferir na infraestrutura de rede dos clientes. Além disso, as medidas de segurança implementadas asseguram a confidencialidade e integridade das conexões, contribuindo para a confiabilidade do sistema.

7.1 Contribuições

O código-fonte deste projeto, incluindo o *script* desenvolvido para execução nas máquinas clientes, a aplicação web em Laravel e as configurações de banco de dados, está armazenado em um repositório público no GitHub. O acesso ao repositório pode ser feito através do link: <https://github.com/Ciccotti/ssh-tunnel>

7.2 Trabalhos Futuros e Perspectivas

Apesar de o sistema desenvolvido atender aos objetivos propostos, há algumas melhorias que podem ser implementadas para aprimorar a usabilidade, segurança e flexibilidade da solução. Algumas direções para trabalhos futuros incluem:

- **Aprimoramento do Armazenamento e Compartilhamento Automático das Chaves SSH**: Atualmente, o armazenamento e compartilhamento das chaves SSH exigem passos manuais. Uma melhoria significativa seria a implementação de um sistema de geração e distribuição automática de chaves SSH diretamente pelo servidor para os clientes autorizados. Esse processo automatizado garantiria que cada cliente receba sua chave de forma segura, simplificando o gerenciamento e reduzindo a necessidade de intervenções manuais para atualizações de chaves;
- **Abertura Simultânea de Múltiplos Túneis**: A interface web de gerenciamento atualmente permite a abertura de um túnel por vez para cada cliente. Uma funcionalidade futura desejável seria a possibilidade de abrir múltiplos túneis simultaneamente para o mesmo cliente, permitindo acessar diferentes serviços ao mesmo tempo;

- **Conversão do *Script* Python em Executável**: A fim de melhorar a segurança e facilitar a distribuição, uma versão compilada do *script* Python pode ser gerada como executável para os sistemas operacionais compatíveis. Isso garantiria maior proteção ao código-fonte, além de facilitar a instalação e execução do *script* pelos usuários.

Essas melhorias representam passos importantes para tornar o sistema ainda mais seguro, prático e adaptável, proporcionando uma melhor experiência para o usuário.

Referências

BARRETT, D. J.; SILVERMAN, R. E.; BYRNES, R. G. **SSH, the Secure Shell: The Definitive Guide**. 2. ed. Sebastopol, CA: O'Reilly Media, Inc., 2005. ISBN 0596008953. Citado na página 12.

CALLAGHAN, S.; JUVE, G.; VAHI, K.; MAECHLING, P.; JORDAN, T.; DEELMAN, E. rvgahp: Push-based job submission using reverse ssh connections. In: **Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science**. ACM, 2017. p. 1–8. Disponível em: <https://dl.acm.org/doi/10.1145/3150994.3151003>. Citado 2 vezes nas páginas 16 e 17.

CLOUDFLARE. **RDP Security Risks**. 2024. Acesso em: 26 nov. 2024. Disponível em: <https://www.cloudflare.com/pt-br/learning/access-management/rdp-security-risks/>. Citado na página 10.

DUSI, M.; GRINGOLI, F.; SALGARELLI, L. A preliminary look at the privacy of ssh tunnels. In: **Proceedings of the 17th International Conference on Computer Communications and Networks (ICCCN)**. IEEE, 2008. p. 1–7. Disponível em: <https://ieeexplore.ieee.org/document/4674282>. Citado na página 13.

GARCIA, M.; PATEL, R. Implementing zero trust architectures in legacy systems. **Computing Security Journal**, v. 12, n. 4, p. 205–220, 2023. Citado na página 10.

HOFFMAN, C. **What Is Reverse SSH Tunneling? (and How to Use It)**. 2019. Acesso em: 11 nov. 2024. Disponível em: <https://www.howtogeek.com/428413/what-is-reverse-ssh-tunneling-and-how-to-use-it/>. Citado na página 13.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. Mqtt-s: A publish/subscribe protocol for wireless sensor networks. In: **Proceedings of the 3rd International Conference on Communication Systems Software and Middleware (COMSWARE '08)**. IEEE, 2008. p. 791–798. Disponível em: <https://ieeexplore.ieee.org/document/4554519>. Citado na página 16.

LEE, M.; CHEN, A. Mitigating unauthorized access through dynamic port management. **International Journal of Network Security**, v. 18, n. 2, p. 87–99, 2021. Citado na página 10.

MAASSEN, J.; BAL, H. E. Smartsockets: Solving the connectivity problems in grid computing. In: **Proceedings of the 16th International Symposium on High Performance Distributed Computing**. ACM, 2007. p. 1–10. Disponível em: <https://doi.org/10.1145/1272366.1272368>. Citado na página 17.

MORENO, D. **Introdução ao Pentest**. São Paulo: Novatec Editora, 2019. Citado 2 vezes nas páginas 9 e 10. Citado 2 vezes nas páginas 9 e 10.

NICOLETTI, P. **Remote Work Security Statistics in 2022**. CyberTalk.org, 2022. Cyber Talk Insights. Acesso em: 12 nov. 2023. Disponível em: <https://www.cybertalk.org/2022/03/31/remote-work-security-statistics-in-2022/>. Citado na página 9.

NOGUEIRA, A. M.; PATINI, A. C. Trabalho remoto e desafios dos gestores. **RAI Revista de Administração e Inovação**, Elsevier, v. 9, n. 4, p. 121–152, 2012. Citado na página 14.

OLIVEIRA, A.; SOUZA, R. O trabalho mudou-se para casa: trabalho remoto no contexto da pandemia de covid-19. **Revista Brasileira de Saúde Ocupacional**, v. 46, n. 4, p. 78–92, 2021. Acesso em: 16 nov. 2024. Disponível em: <https://www.scielo.br/j/rbso/a/LQnfJLrjgrSDKkTNyVfgnQy/>. Citado na página 14.

PAN, X.; HU, H.; XU, J.; LI, M. Research on video surveillance robot based on ssh reverse tunnel technology. In: **Proceedings of the 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)**. IEEE, 2020. p. 298–302. Acesso em: 26 nov. 2024. Disponível em: <https://doi.org/10.1109/AEMCSE50948.2020.00071>. Citado na página 17.

SARAMAGO, M. V. M. **Desenvolvimento de uma aplicação web para acesso remoto a computadores de bordo utilizados no ramo da automação agrícola**. Trabalho de Conclusão de Curso de Graduação — Universidade Federal de Santa Catarina, Florianópolis, 2020. Acesso em: 8 fev. 2023. Disponível em: <https://repositorio.ufsc.br/handle/123456789/243609>. Citado 2 vezes nas páginas 9 e 17.

SHEWALE, R. **Internet User Statistics In 2023: Global Demographics**. DemandSage, 2023. Acesso em: 8 nov. 2023. Disponível em: <https://www.demandsage.com/internet-user-statistics/>. Citado na página 9.

SILVA, J.; SANTOS, M. O trabalho remoto/home-office no contexto da pandemia covid-19. **Revista de Economia e Gestão**, v. 25, n. 2, p. 45–58, 2020. Acesso em: 26 nov. 2024. Disponível em: <https://www3.eco.unicamp.br/remir/images/Artigos_2020/ARTIGO_REMIR.pdf>. Citado na página 15.

SMITH, J.; DOE, J. Secure remote access without firewall modifications. **Journal of Cybersecurity and Network Management**, v. 15, n. 3, p. 123–134, 2022. Citado 2 vezes nas páginas 10 e 13.

SOUPPAYA, M.; SCARFONE, K. **Guide to Enterprise Telework, Remote Access, and Bring Your Own Device (BYOD) Security**. Gaithersburg, MD, 2016. Disponível em: <https://doi.org/10.6028/NIST.SP.800-46r2>. Citado na página 15.

STATS, I. W. **World Internet Usage and Population Statistics**. 2022. Acesso em: 8 fev. 2023. Disponível em: <https://www.internetworldstats.com/stats.htm>. Citado na página 9.

TOLEDO, S. Ssh tunneling to connect to remote computers. **Software Impacts**, v. 17, p. 100545, October 2023. ISSN 2665-9638. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2665963823000829>. Citado 2 vezes nas páginas 15 e 16.

WAZLAWICK, R. S. **Metodologia de Pesquisa para Ciência da Computação**. 2. ed. Rio de Janeiro, RJ, Brasil: Elsevier, 2009. Acesso em: 11 nov. 2024. ISBN 9788535266436. Disponível em: <http://books.google.com.br/books?id=ZLbCKKWQ6OQC>. Citado na página 19.

YLONEN, T.; TURNER, P.; SCARFONE, K.; SOUPPAYA, M. **Security of Interactive and Automated Access Management Using Secure Shell (SSH)**. Gaithersburg, MD, EUA, 2015. Disponível em: <https://doi.org/10.6028/NIST.IR.7966>. Citado na página 12.