UNIVERSIDADE ESTADUAL DO MARANHÃO

CENTRO DE CIÊNCIAS TECNOLÓGICAS

CURSO DE ENGENHARIA DA COMPUTAÇÃO

ERICK FERREIRA SOUSA

SISTEMA ESCALÁVEL DE RESPOSTA ATIVA PARA MITIGAÇÃO DE ATAQUES DE FORÇA BRUTA EM ACTIVE DIRECTORY COM SURICATA

São Luís

2025

ERICK FERREIRA SOUSA

SISTEMA ESCALÁVEL DE RESPOSTA ATIVA PARA MITIGAÇÃO DE ATAQUES DE FORÇA BRUTA EM ACTIVE DIRECTORY COM SURICATA

Monografia apresentada ao Curso de Engenharia da Computação da Universidade Estadual do Maranhão para obtenção do grau de bacharelado em Engenharia de Computação.

Orientador: Prof. Dr. Wesley Dominices Batista de Araujo

São Luís

2025

Sousa, Erick Ferreira.
Sistema escalável de resposta ativa para mitigação de ataques de força bruta em active directory com suricata / Erick Ferreira Sousa. - São Luís - MA, 2025.
88 f.

Monografia (Graduação em Engenharia da Computação) - Universidade Estadual do Maranhão, São Luís, 2025.

Orientador: Prof. Dr. Wesley Batista Dominices de Araújo.

1. Segurança. 2. Defesa. 3. Prevenção. 4. Intrusões. I. Título.

CDU: 004.056.53

Elaborado por Luciana de Araújo - CRB 13/445

ERICK FERREIRA SOUSA

SISTEMA ESCALÁVEL DE RESPOSTA ATIVA PARA MITIGAÇÃO DE ATAQUES DE FORÇA BRUTA EM ACTIVE DIRECTORY COM SURICATA

Monografia apresentada ao Curso de Engenharia da Computação da Universidade Estadual do Maranhão para o grau de bacharelado em Engenharia de Computação.

Aprovado em:18/12/2025

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br
WESLEY BATISTA DOMINICES DE ARAUJO
Data: 06/01/2026 12:15:20-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Wesley Batista Dominices de Araújo

Doutor em Engenharia Elétrica

Universidade Federal do Maranhão

Documento assinado digitalmente
gov.br
PEDRO BRANDAO NETO
Data: 07/01/2026 08:35:55-0300
Verifique em https://validar.iti.gov.br

Prof. Me. Pedro Brandão Neto

Mestre em Engenharia Elétrica

Universidade Federal do Maranhão

Documento assinado digitalmente
gov.br
MARCOS JOSE DOS PASSOS SA
Data: 08/01/2026 09:43:07-0300
Verifique em https://validar.iti.gov.br

Prof. Me. Marcos José dos Passos Sá

Mestre em Engenharia da Computação e Sistemas

Universidade Estadual do Maranhão

Dedico à minha vó e à minha mãe, que começaram esta luta há anos atrás para que eu tivesse a oportunidade de estudar e me graduar.

RESUMO

Este trabalho aborda a análise e o desenvolvimento de um sistema de resposta ativa para a mitigação de ataques de força bruta direcionados ao serviço de controlador de domínio Active Directory. A crescente sofisticação das ameaças cibernéticas demanda soluções que não apenas detectem, mas também neutralizem ataques em tempo real. A metodologia empregada consistiu na construção de um ambiente virtualizado, composto por uma máquina de ataque (Kali Linux), um alvo (Windows Server 2022 com Active Directory) e um sistema de defesa (Ubuntu Server) posicionado em uma topologia de rede *in-line*, atuando como um Sistema de Prevenção de Intrusões — *Intrusion Prevention System* (IPS). A detecção da ameaça foi realizada pelo motor de segurança Suricata, configurado com uma regra personalizada para identificar o tráfego anômalo característico do ataque via protocolo *Server Message Block* (SMB). A mitigação foi automatizada por meio de um script em Bash que, ao ser acionado pelos alertas do Suricata, executava o bloqueio dinâmico do endereço de *Internet Protocol* (IP) do atacante no firewall do sistema. Os resultados validaram a eficácia do protótipo, demonstrando sua capacidade de detectar e bloquear com sucesso o ataque simulado, impedindo a continuidade da atividade maliciosa. Conclui-se que a arquitetura proposta é uma solução viável, escalável e economicamente acessível para a proteção de infraestruturas críticas, estabelecendo um arcabouço funcional que pode ser expandido para mitigar outras ameaças e servir como alternativa robusta a soluções comerciais de alto custo.

Palavras-chave: Segurança da Informação. Active Directory. Sistema de Prevenção de Intrusões. Suricata. Resposta Ativa.

ABSTRACT

This work addresses the development and analysis of an active response system for mitigating brute-force attacks targeting the Active Directory domain controller service. The growing sophistication of cyber threats demands solutions that not only detect but also neutralize attacks in real time. The methodology employed consisted of building a virtualized lab environment featuring an attacking machine (Kali Linux), a target (Windows Server 2022 with Active Directory), and a defense system (Ubuntu Server) positioned in an in-line network topology to act as an Intrusion Prevention System (IPS). Threat detection was performed by the Suricata security engine, configured with a custom rule to identify anomalous traffic characteristic of the attack via the Service Message Block (SMB) protocol. Mitigation was automated through a Bash script, which, when triggered by Suricata alerts, dynamically blocked the attacker's Internet Protocol (IP) address in the system firewall. The results validated the prototype's effectiveness, demonstrating its ability to successfully detect and block the simulated attack, thereby preventing the continuation of malicious activity. It is concluded that the proposed architecture is a viable, scalable, and cost-effective solution for protecting critical infrastructures, establishing a functional framework that can be expanded to mitigate other threats and serve as a robust alternative to high-cost commercial solutions.

Keywords: Information Security. Active Directory. Intrusion Prevention System. Suricata. Active Response.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

AD – Active Directory

AD DS – Active Directory Domain Services

AMP – Advanced Malware Protection

API – Application Programming Interface

AS-REP – Authentication Service Response

ATP – Advanced Threat Protection

CeWL – Custom Word List generator

CID – Confidencialidade, Integridade e Disponibilidade

CME – CrackMapExec

CUPP – Common User Password Profiler

CVE – Common Vulnerabilities and Exposures

DDoS – Distributed Denial of Service

DHCP – Dynamic Host Configuration Protocol

DMZ – Demilitarized Zone

DNS – Domain Name System

DoS – Denial of Service

DPDK – Data Plane Development Kit

DPI – Deep Packet Inspection

ELK – Elasticsearch Logstash Kibana

FTP – File Transfer Protocol

GPO – Group Policy Object

GPL – General Public License

HTTP – Hypertext Transfer Protocol

ICMP – Internet Control Message Protocol

IDS – Intrusion Detection System

IoT – Internet of Things

IP – Internet Protocol

IPS – Intrusion Prevention System

JSON – JavaScript Object Notation

JtR – John the Ripper

LGPD – Lei Geral de Proteção de Dados

MQTT – Message Queuing Telemetry Transport

NAT – Network Address Translation

NGFW – Next-Generation Firewalls

NIC – Network Interface Card

OISF – Open Information Security Foundation

PME – Pequenas e Médias Empresas

RAM – Random Access Memory

RDP – Remote Desktop Protocol

ROI – Return on Investment

RSS – Receive Side Scaling

SID – Signature ID

SIEM – Security Information and Event Management

SMB – Server Message Block

SOAR – Security Orchestration, Automation and Response

SQL – Structured Query Language

SSH – Secure Shell

TCO – Total Cost of Ownership

TCP – Transmission Control Protocol

TLS – Transport Layer Security

UDP – User Datagram Protocol

UFW – Uncomplicated Firewall

UTM – Unified Threat Management

VBS – Virtualization-Based Security

VLAN – Virtual Local Area Network

VMs – Virtual Machines

WEP – Wired Equivalent Privacy

WinRM – Windows Remote Management

WPA – Wi-Fi Protected Access

SUMÁRIO

1 Introdução

A segurança da informação tornou-se um dos pilares fundamentais da sociedade digital contemporânea, sustentando as operações de organizações públicas e privadas que dependem cada vez mais de sistemas informatizados e conectividade constante. Em um contexto onde praticamente todos os serviços essenciais estão interligados desde o armazenamento em nuvem até a autenticação de identidades corporativas, a proteção de dados e de infraestruturas críticas assume papel estratégico. O avanço das tecnologias de informação, acompanhado pela expansão do ambiente cibernético, trouxe não apenas benefícios, mas também uma ampliação significativa do campo de ameaças digitais. Nesse cenário, ataques automatizados, exploração de vulnerabilidades e campanhas de engenharia social tornaram-se cada vez mais sofisticados, exigindo soluções que unam capacidade de detecção, resposta e resiliência.

Entre os alvos mais sensíveis de ambientes corporativos destaca-se o Active Directory (AD), tecnologia amplamente utilizada para a administração centralizada de usuários, políticas e permissões em redes baseadas em Windows. Por concentrar funções críticas de autenticação e controle de acesso, o AD é um dos componentes mais visados por atacantes. O comprometimento desse serviço pode resultar no controle total de uma rede corporativa, possibilitando o roubo de credenciais, a manipulação de políticas de segurança e o acesso indevido a informações confidenciais. Dentre as diversas técnicas empregadas para comprometer o AD, o ataque de força bruta é uma das mais recorrentes e diretas, explorando falhas de autenticação e credenciais fracas.

Ataques de força bruta consistem na tentativa automatizada e repetitiva de múltiplas combinações de nomes de usuários e senhas, até que a correta seja identificada. Apesar de simples, essa técnica se mantém altamente efetiva, especialmente em ambientes que carecem de políticas de bloqueio de conta, autenticação multifator ou sistemas de monitoramento. Em muitas redes, especialmente de pequeno e médio porte, o processo de autenticação do AD continua exposto a esse tipo de ameaça, e sua detecção depende exclusivamente de logs ou auditorias manuais, frequentemente analisadas apenas após a ocorrência de um incidente. Essa defasagem entre a detecção e a mitigação representa uma das fragilidades mais críticas em arquiteturas de segurança corporativa.

Um Sistema de Detecção de Intrusões tradicional atua de forma passiva, identificando atividades suspeitas e gerando alertas. No entanto, a resposta a tais alertas depende de ação humana, o que pode resultar em atraso na contenção de ameaças. Em contraste, um Sistema de

Prevenção de Intrusões adiciona uma camada ativa de defesa, capaz de bloquear automaticamente tentativas maliciosas, minimizando o tempo de exposição do sistema. Essa capacidade de resposta imediata é essencial em ataques automatizados, como os de força bruta, em que milhares de tentativas podem ocorrer em poucos segundos.

Para materializar essa capacidade de prevenção, a metodologia deste trabalho seguiu um percurso experimental evolutivo. Inicialmente, o sistema de defesa foi implementado em uma topologia *out-of-band*, onde o Ubuntu Server com Suricata atuava como um sensor passivo, apenas detectando e alertando sobre o tráfego do ataque. Posteriormente, a arquitetura foi deliberadamente reconfigurada para um modelo *in-line*, posicionando o servidor de defesa como um gateway obrigatório. Essa transição foi fundamental para demonstrar na prática a diferença conceitual e funcional entre a simples detecção de um IDS e a capacidade de bloqueio preventivo de um IPS, que é o cerne da solução proposta.

O presente trabalho propõe o desenvolvimento de um sistema de resposta ativa capaz de mitigar ataques de força bruta direcionados ao Active Directory, combinando o uso do Suricata que é um motor de análise de tráfego e detecção de intrusões de código aberto. Juntamente com scripts automatizados executados em Ubuntu Server, a solução foi projetada para atuar de forma *in-line*, interceptando o tráfego entre o atacante e o servidor de domínio, analisando pacotes em tempo real e, quando identificado um padrão de ataque, acionando um bloqueio dinâmico no firewall do sistema.

Os resultados experimentais validaram a eficácia do protótipo, que se mostrou capaz de neutralizar o ataque em poucos instantes após sua detecção. A análise da mitigação confirmou que, uma vez acionado o script de resposta, o endereço IP do atacante é imediatamente bloqueado no firewall do gateway, resultando em um erro de tempo de conexão excedido na máquina atacante. Esta validação prática demonstrou que o sistema não apenas cumpre seu objetivo de bloquear a ameaça, mas também impede tentativas subsequentes, garantindo uma proteção persistente contra a origem do ataque.

Além da validação funcional, o estudo aprofundou-se na análise de desempenho das tecnologias de automação empregadas. Foram desenvolvidos e comparados dois protótipos para o script de resposta ativa, um em Python e outro em Bash, a fim de determinar a abordagem mais eficiente para um dispositivo de segurança de rede. A escolha não se baseou apenas na funcionalidade, mas em critérios de consumo de recursos e latência, que são fatores críticos em um ambiente de gateway, onde o desempenho do tráfego legítimo não pode ser comprometido.

Para quantificar essa eficiência, foi realizada uma simulação de um cenário de alta carga, com centenas de usuários realizando requisições de autenticação simultâneas. Os dados coletados revelaram que, embora ambas as soluções sejam funcionais, o script em Bash impõe uma penalidade de desempenho significativamente menor. Este resultado forneceu uma justificativa técnica robusta para a adoção do Bash como a solução final, reforçando o compromisso do projeto com a otimização e a viabilidade prática em ambientes de produção.

Além de sua importância técnica, este trabalho apresenta relevância prática e econômica. Soluções comerciais de prevenção de intrusões, como Next-Generation Firewalls (NGFW) ou sistemas de segurança gerenciada, costumam envolver altos custos de aquisição e manutenção, inviabilizando sua adoção por pequenas e médias empresas. A proposta aqui desenvolvida demonstra que, por meio de ferramentas *open source*, é possível construir um sistema funcional, escalável e de baixo custo, sem depender de licenças proprietárias ou contratos de suporte dispendiosos. Essa abordagem democratiza o acesso à segurança cibernética avançada, tornando-a acessível a diferentes contextos organizacionais e acadêmicos.

Do ponto de vista acadêmico, este projeto reforça a importância da integração entre o conhecimento teórico e a prática experimental na formação de profissionais de engenharia e cibersegurança. A implementação de um ambiente virtualizado, envolvendo máquinas com Kali Linux, Windows Server 2022 e Ubuntu Server, permitiu a simulação de cenários reais de ataque e defesa em uma rede corporativa. Esse método experimental oferece uma visão clara de como ataques de autenticação se propagam, como são detectados em nível de pacote e quais estratégias são mais eficazes para mitigá-los em tempo real.

A partir de um incidente real que serviu de base para este estudo, no qual a suspensão temporária de políticas de bloqueio de conta resultou em um ataque de força bruta bem-sucedido, observou-se a necessidade de um mecanismo capaz de responder automaticamente a ameaças sem intervenção manual. Esse contexto motivou o desenvolvimento de um protótipo de defesa proativa, que não apenas detecta, mas também reage ao comportamento anômalo, interrompendo o ataque antes que a integridade da rede seja comprometida.

Além da mitigação técnica, o estudo propõe reflexões sobre a importância do monitoramento contínuo e da correlação de eventos de segurança para a manutenção da integridade e disponibilidade de sistemas corporativos. A ausência de mecanismos automatizados de análise e resposta pode resultar em impactos severos, como interrupção de serviços críticos, roubo de informações sensíveis e prejuízos financeiros. Por outro lado, a

adoção de soluções inteligentes e economicamente viáveis contribui para o fortalecimento da postura de segurança, atendendo também a requisitos de conformidade com regulamentações como a Lei Geral de Proteção de Dados (LGPD).

Assim, este trabalho busca não apenas apresentar uma solução técnica funcional, mas também evidenciar a importância da resposta ativa e automatizada como tendência na cibersegurança moderna. Com a crescente sofisticação dos ataques e a redução do tempo médio entre a detecção e o comprometimento, a automação da defesa torna-se uma necessidade inevitável. O sistema desenvolvido, fundamentado em software livre e em práticas consolidadas de segurança de rede, representa uma alternativa prática, escalável e acessível para ambientes corporativos e institucionais.

Dessa forma, a pesquisa contribui tanto para o campo acadêmico quanto para o profissional, oferecendo um modelo de infraestrutura de defesa ativa que pode ser replicado, aprimorado e adaptado a diferentes contextos. Ao longo dos capítulos seguintes, são detalhados: o embasamento teórico, a metodologia experimental, a implementação da arquitetura proposta e a análise dos resultados obtidos, com ênfase na eficácia e no custo-benefício do sistema desenvolvido em comparação com soluções comerciais de alto investimento.

1.1 Objetivo Geral

Desenvolver e analisar a eficácia de um sistema de resposta ativa para a detecção e mitigação em tempo real de ataques de força bruta por dicionário, direcionados ao serviço Active Directory em um ambiente de rede simulado.

1.2 Objetivos Específicos

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

- **Construir** um ambiente de laboratório virtualizado e isolado, utilizando o Oracle VirtualBox, contendo uma máquina atacante (Kali Linux), um alvo (Windows Server 2022 com Active Directory) e um sistema de defesa (Ubuntu Server).
- **Configurar** o motor de detecção Suricata no Ubuntu Server com uma regra personalizada para identificar especificamente o tráfego característico de um ataque de força bruta via protocolo SMB.
- **Desenvolver** um script de resposta ativa em Bash que monitore os logs de alerta do Suricata e execute automaticamente o bloqueio do endereço IP do atacante no firewall do Ubuntu Server.

- **Validar** a eficácia do sistema através da execução de um ataque simulado e da análise dos resultados de detecção e bloqueio, confirmando a mitigação da ameaça.

1.3 Justificativa

A realização deste projeto é justificada pela crescente necessidade de soluções de segurança eficazes e acessíveis para proteger infraestruturas de rede contra ameaças persistentes. O estudo foi inspirado por um incidente de segurança real, no qual a desativação temporária de políticas de bloqueio de conta para resolver um problema de acesso legítimo expôs um utilizador do Active Directory a um ataque de força bruta bem-sucedido, resultando no comprometimento de suas credenciais. Este evento real destaca a relevância prática de se ter uma camada de defesa de rede que possa atuar independentemente das políticas do sistema operacional alvo.

Do ponto de vista tecnológico e financeiro, o projeto demonstra a viabilidade de construir um Sistema de Prevenção de Intrusão robusto utilizando exclusivamente ferramentas de código aberto, como Ubuntu Server com o Suricata. Esta abordagem representa uma alternativa ou complemento de baixo custo a soluções comerciais dispendiosas, como Firewalls de Próxima Geração (NGFW), tornando a segurança ativa acessível a pequenas e médias empresas que não dispõem de grandes orçamentos para cibersegurança.

Este trabalho contribui academicamente abordando um estudo prático e detalhado sobre a implementação de uma arquitetura de segurança *in-line*, a criação de regras de detecção personalizadas e o desenvolvimento de um mecanismo de resposta automatizada. A metodologia experimental, realizada em um ambiente de laboratório seguro, permitiu não apenas validar a solução, mas também explorar e documentar os desafios técnicos e as melhores práticas envolvidas, oferecendo um material didático relevante para estudantes e profissionais da área de segurança da informação.

2 Revisão Teórica

2.1 Fundamentos de Segurança Cibernética

A cibersegurança é o campo responsável por proteger sistemas computacionais, redes e dados contra acessos não autorizados, ataques e falhas que possam comprometer a integridade, a confidencialidade ou a disponibilidade de dados. Para compreender os fundamentos dessa área, é necessário conhecer quatro conceitos essenciais: ameaça, vulnerabilidade, risco e incidente de segurança (STAMFORD, 2020).

A ameaça é definida como qualquer circunstância ou agente com potencial para causar danos a um sistema de informação por meio de ações acidentais ou intencionais. Exemplos comuns são softwares maliciosos (malware), ataques de força bruta, phishing, engenharia social, falhas humanas ou até desastres naturais que afetam recursos físicos (TANENBAUM; WETHERALL, 2011). Em ambientes residenciais, uma ameaça recorrente é a tentativa de invasão de redes Wi-Fi por cibercriminosos.

A vulnerabilidade corresponde a uma fraqueza presente em um ativo, sistema ou processo, que pode ser explorada por uma ameaça. Exemplos incluem o uso de senhas fracas, falhas não corrigidas em softwares, dispositivos desatualizados ou configurações incorretas em roteadores domésticos (CERT.BR, 2023). A existência de uma vulnerabilidade isoladamente não representa um ataque, mas sim a possibilidade de que ele ocorra, um potencial problema.

O risco é o resultado da interação entre ameaças e vulnerabilidades. Pode ser entendido como a probabilidade de uma ameaça explorar uma vulnerabilidade e causar um impacto negativo. Em outras palavras, risco é a medida da exposição a potenciais perdas e pode ser mitigado por meio da aplicação de controles de segurança, como autenticação, criptografia e políticas de atualização periódica (SÊMOLA 2014).

Um incidente de segurança ocorre quando uma ameaça se concretiza e explora uma vulnerabilidade provocando um dado real, como a interrupção de serviços, o roubo de informações ou o comprometimento de dispositivos (BARROS, 2021). Incidentes em redes residenciais incluem o vazamento de senhas, o sequestro de dados por ransomware ou a espionagem por câmeras IP invadidas.

2.1.1 Princípios de Segurança da Informação

A parte crucial da proteção informacional reside nos alicerces do CID: Confidencialidade, Integridade e Disponibilidade. Tais fundamentos moldam cada norma, ação ou ferramenta focada na salvaguarda de dados e sistemas, seja no âmbito empresarial, estatal ou familiar. Empresas de todo o tipo usam estes pilares como guia no trato de informações (SÊMOLA, 2014).

A confidencialidade se concentra na blindagem dos dados contra acessos proibidos. O intuito é certificar que só pessoas, dispositivos ou plataformas com permissão possam ver ou usar certos dados. Tal preceito é vital em situações que envolvem dados sensíveis, como palavras-chave, dados bancários, mensagens ou fotos tiradas por câmeras em casas conectadas à web (BARROS, 2021).

A violação da confidencialidade pode ocorrer via espionagem, captura de dados em redes Wi-Fi abertas ou abuso de falhas em dispositivos da Internet das Coisas (IoT), o que reforça a relevância de métodos como criptografia, autenticação forte e divisão de redes (CERT.BR, 2023).

A integridade assegura que os dados não sofram mudanças indevidas, seja por erro ou dolo, durante seu armazenamento, uso ou tráfego. Assim, os dados devem manter-se fiéis e precisos do início ao fim. Em lares, por exemplo, é crucial garantir que arquivos em servidores domésticos, como vídeos ou textos, não sejam danificados ou corrompidos por vírus (STAMFORD, 2020).

A disponibilidade garante que os dados estejam acessíveis e prontos para uso sempre que preciso por pessoas ou sistemas com permissão. A perda de acesso pode ocorrer por falhas, falta de luz, ou ataques como os de negação de serviço (Denial of Service — DoS/ Distributed Denial of Service — DDoS), que são comuns até em casas, em especial em dispositivos online (OLIVEIRA, 2022).

2.1.2 Responsabilidades Organizacionais Sobre Dados

Devido à crescente presença digital nos serviços e ao uso intensivo das tecnologias de informação, as empresas passaram a reunir, tratar e guardar grandes quantidades de informações pessoais de seus utilizadores. Assim, nasce para essas instituições uma obrigação ética, jurídica e técnica no que se refere à segurança, à privacidade e ao uso correto desses dados (SÊMOLA, 2014).

A obrigação jurídica ganhou relevância com a promulgação da Lei Geral de Proteção de Dados no Brasil (Lei nº 13.709/2018), que define os princípios, direitos e deveres relacionados ao tratamento de dados pessoais. Segundo a LGPD, cada empresa que recolhe dados de pessoas físicas, ainda que seja apenas o nome e e-mail, deve assegurar que esse tratamento aconteça de maneira clara, com consentimento explícito e de acordo com os princípios da finalidade, necessidade e segurança (BRASIL, 2018).

Entre as principais obrigações jurídicas das empresas, merecem destaque:

- Informar de forma clara ao utilizador quais os dados que estão a ser recolhidos e com que objetivo.
- Assegurar mecanismos de autorização e permitir que o utilizador retire essa autorização a qualquer momento.
- Adotar medidas técnicas e administrativas para proteger os dados contra acessos indevidos, vazamentos ou destruição acidental (BARROS, 2021).

Além do aspecto jurídico, existe também uma obrigação ética, ligada à confiança que o utilizador deposita na empresa ao fornecer os seus dados. Uma fuga de informações pode causar não só prejuízos financeiros e jurídicos à empresa, mas também danos graves à reputação e perda de credibilidade perante os seus clientes e parceiros (STAMFORD, 2020).

Do ponto de vista técnico, é dever da empresa implementar políticas de segurança da informação, como controle de acesso, encriptação, cópias de segurança, atualização de sistemas e formações internas. Estas práticas compõem um programa de gestão de dados, crucial para minimizar riscos e responder de forma adequada a incidentes de segurança.

2.1.3 Arquiteturas de Defesa de Rede, Perímetro e DMZ

Em ambientes corporativos, a segurança de redes é tradicionalmente estruturada a partir do conceito de perímetro de rede, que representa a fronteira lógica e física entre a rede interna da organização (confiável) e redes externas não confiáveis, como a Internet. Esse perímetro é protegido por mecanismos de controle de acesso, monitoramento e filtragem de tráfego, com o objetivo de reduzir a superfície de ataque e impedir acessos não autorizados aos ativos internos (BARROS, 2021).

Como parte fundamental dessa arquitetura, destaca-se a Zona Desmilitarizada (DMZ – *Demilitarized Zone*), definida como uma sub-rede intermediária posicionada entre a rede externa e a rede interna. A DMZ tem como finalidade permitir a exposição controlada de

serviços que necessitam ser acessados externamente, como servidores web, servidores de correio eletrônico, servidores de aplicações públicas e gateways de acesso remoto, sem que esses serviços tenham acesso direto à rede interna (CERT.BR, 2023).

Do ponto de vista arquitetural, a DMZ funciona como uma camada de isolamento, reduzindo o impacto de possíveis comprometimentos. Caso um serviço hospedado na DMZ seja explorado por um atacante, os mecanismos de filtragem e segmentação impedem, ou ao menos dificultam significativamente, o avanço do ataque para a rede interna. Essa separação é normalmente implementada por meio de firewalls, VLANs ou roteamento controlado, com regras específicas para cada zona de segurança (TANENBAUM; WETHERALL, 2011).

Uma característica essencial da DMZ é o controle rigoroso do fluxo de tráfego. Em geral, conexões iniciadas a partir da Internet são permitidas apenas para serviços específicos na DMZ, enquanto o acesso da DMZ para a rede interna é extremamente restrito ou inexistente. Da mesma forma, a comunicação da rede interna para a DMZ ocorre apenas quando estritamente necessária, seguindo o princípio do menor privilégio (*least privilege*) (CERT.BR, 2023).

É importante destacar que serviços críticos de autenticação, autorização e gerenciamento de identidade, como diretórios corporativos, não devem ser posicionados na DMZ. Esses sistemas concentram informações sensíveis e desempenham papel central na segurança da organização, devendo permanecer em segmentos internos altamente protegidos. A DMZ, portanto, não substitui mecanismos de segurança internos, mas atua como uma camada adicional dentro de uma estratégia mais ampla de defesa em profundidade (BARROS, 2021).

Dessa forma, a adoção de uma DMZ bem estruturada contribui para a redução de riscos, melhora a segmentação da rede e estabelece uma base sólida para a implementação de controles de segurança adicionais, sendo amplamente reconhecida como uma boa prática em arquiteturas de redes seguras.

2.2 Ataques de Força Bruta

Os ataques de força bruta são uma das formas mais comuns e diretas de quebra de autenticação digital. Eles consistem na tentativa sistemática de adivinhação de senhas ou chaves criptográficas, por meio da repetição automática de combinações até que a correta seja descoberta (BARROS, 2021). Esses ataques são amplamente utilizados por agentes maliciosos contra contas de e-mail, redes Wi-Fi, sistemas de login e dispositivos conectados em redes residenciais, especialmente quando medidas de proteção básicas não foram aplicadas.

Ao contrário de técnicas mais sofisticadas que exigem engenharia social ou exploração de vulnerabilidades, o ataque de força bruta depende de **capacidade computacional** e **persistência**. Quanto mais fraca for a senha e menor for a complexidade do sistema de autenticação, maior será a chance de sucesso do ataque (STAMFORD, 2020).

No ataque por dicionário, o invasor utiliza uma lista de palavras pré-definidas (geralmente palavras comuns, nomes próprios, datas ou senhas vazadas anteriormente) para tentar acesso. Essa técnica parte do princípio de que muitos usuários escolhem senhas fracas ou previsíveis, como "123456", "admin" ou "senha123".

Esse método é mais rápido que o exaustivo, mas limitado ao conteúdo do dicionário. Ainda assim, é bastante eficiente contra usuários leigos, bastante utilizado em redes domésticas e dispositivos IoT com senhas padrão, porém, também é usado após um ataque de engenharia social ou phishing onde o atacante consegue informações preliminares sobre senhas de um alvo (CERT.BR, 2023).

No ataque exaustivo, o invasor testa todas as combinações possíveis de caracteres, iniciando geralmente pelas senhas mais curtas até atingir a correta. Essa técnica garante sucesso eventualmente, mas pode ser extremamente demorada se a senha for longa e complexa.

Esse tipo de ataque é mais eficaz contra sistemas sem bloqueio de tentativas ou limitação de tempo entre os acessos, e é frequentemente usado em redes Wi-Fi ou sistemas criptográficos (TANENBAUM; WETHERALL, 2011).

O ataque híbrido combina as estratégias de dicionário e exaustiva. Nesse modelo, o atacante usa palavras do dicionário, mas realiza variações combinando números, letras maiúsculas e símbolos. Por exemplo, se a palavra do dicionário for "senha", o ataque tentará variações como "Senha1", "s3nh@" ou "senha2024". Esse método equilibra **eficiência** e **abrangência**, sendo mais eficaz que o dicionário puro e menos custoso que o exaustivo completo (SÊMOLA, 2014).

2.3 Exemplos Históricos

- **Ataque ao Yahoo (2012–2014):** Entre 2012 e 2014, o Yahoo sofreu um dos maiores vazamentos de dados da história, que envolveu o roubo de informações de mais de 3 bilhões de contas. Embora o ataque tenha usado múltiplas técnicas, investigações apontaram que a violação inicial envolveu força bruta e exploração de credenciais fracas, incluindo o uso de senhas repetidas em múltiplos serviços (BARROS, 2021).

- **Ataques de Credential Stuffing contra Netflix, Spotify e Zoom (2019–2021):** Com o aumento de vazamentos em massa de senhas na dark web, criminosos passaram a utilizar técnicas automatizadas de força bruta combinadas com ataques de *credential stuffing* (uso de senhas vazadas para acessar outros serviços). Diversos usuários tiveram suas contas em plataformas populares invadidas por causa do reuso de senhas simples e da ausência de autenticação em dois fatores (STAMFORD, 2020).
- **Invasão à Apple iCloud (2014 – "Celebgate"):** Outro exemplo notório ocorreu em 2014, quando diversas contas do iCloud de celebridades foram invadidas. As senhas foram descobertas por meio de ataques de força bruta por dicionário, utilizando senhas fracas como "password" e "123456". O incidente expôs fotos e dados pessoais e levantou debates sobre a necessidade de autenticação multifator em serviços de nuvem (TANENBAUM; WETHERALL, 2011).
- **Ataques a Dispositivos IoT Residenciais:** Diversos estudos acadêmicos e relatórios de segurança têm mostrado que dispositivos IoT (como câmeras, lâmpadas inteligentes e assistentes virtuais) são frequentemente invadidos por meio de ataques de força bruta utilizando senhas padrão de fábrica. Um exemplo amplamente citado é o caso da botnet Mirai (2016), que explorava dispositivos IoT com credenciais simples para realizar ataques DDoS massivos, afetando até grandes provedores como DynDNS (OLIVEIRA, 2022).

2.4 Kali Linux

O Kali Linux é uma distribuição do Linux voltada especificamente para testes de invasão (pentest), análise forense e auditoria de segurança da informação. Desenvolvido e mantido pela equipe da Offensive Security, o Kali Linux é amplamente utilizado tanto por profissionais de cibersegurança quanto por entusiastas e pesquisadores da área (BARROS, 2021).

O sistema vem com centenas de ferramentas de segurança pré-instaladas, que permitem a realização de varreduras, análise de vulnerabilidades, exploração de sistemas, interceptação de tráfego, quebra de senhas, clonagem de redes Wi-Fi e outras atividades que simulam ataques cibernéticos reais. Seu uso é legal e ético quando aplicado em ambientes controlados ou autorizados, mas também pode ser explorado por agentes maliciosos para realizar invasões e comprometer sistemas (STAMFORD, 2020).

Entre as ferramentas mais utilizadas no Kali Linux, destacam-se:

- **Nmap**: para varredura de portas e identificação de serviços em rede.

- **Aircrack-ng**: para ataques a redes Wi-Fi, incluindo captura de handshakes e quebra de senhas WEP/WPA.
- **Metasploit Framework**: um dos mais poderosos ambientes de exploração de vulnerabilidades.
- **Hydra**: ferramenta de ataque por força bruta a serviços como *Secure Shell* (SSH), *File Transfer Protocol* (FTP), *HiperText Transfer Protocol* (HTTP) e *Telnet*.
- **CrackMapExec**: Uma alternativa e mais moderna do que o Hydra, especializada para ambientes Windows e Active Directory, ambas têm o mesmo objetivo – automatizar tentativas de login em massa a partir de listas de possíveis credenciais.
- **Wireshark**: para análise de tráfego e interceptação de pacotes.
- **John the Ripper**: para quebra de senhas com dicionários e força bruta.

Essas ferramentas, quando mal utilizadas, podem comprometer totalmente a segurança de uma rede residencial ou corporativa.

A utilização do Kali Linux no campo da segurança ofensiv**a** é parte fundamental de processos de auditoria, pois permite testar a resistência de redes e sistemas frente a ataques reais, sem que danos reais ocorram. Entretanto, as mesmas ferramentas que servem para proteger, quando nas mãos erradas, tornam-se um poderoso arsenal para atacar redes desprotegidas, roubar dados ou derrubar serviços (SÊMOLA, 2014).

A facilidade de acesso ao Kali Linux (que é gratuito e de código aberto), somada à grande quantidade de tutoriais disponíveis na internet, torna-o um vetor comum de aprendizado para atacantes iniciantes, conhecidos como *script kiddies*, que utilizam ferramentas prontas sem conhecimento profundo, mas com potencial de causar danos reais (TANENBAUM; WETHERALL, 2011).

2.4.1 CrackMapExec

O CrackMapExec (CME) é uma ferramenta de segurança cibernética de código aberto, muito popular entre profissionais de segurança, pentester e red teamers. Ele é usado para automatizar a avaliação de segurança de grandes redes Windows, principalmente ambientes com o Active Directory. Ele não é apenas uma ferramenta de força bruta; é uma plataforma completa que combina várias funcionalidades em um único pacote.

O CME opera sobre o protocolo SMB (Server Message Block), mas também pode interagir com outros protocolos como o Windows Remote Management (WinRM) e o Kerberos. As principais funcionalidades incluem:

- **Enumeração de Rede:** O CME pode escanear grandes redes em busca de máquinas Windows ativas, listar usuários, grupos, compartilhamentos de rede, sessões ativas e até mesmo senhas do Active Directory (se houver vulnerabilidades como o Authentication Service Response — AS-REP). Isso ajuda a mapear a rede e entender sua topologia de forma rápida e eficiente.
- **Ataques de Autenticação:** Esta é a principal função. O CME pode executar ataques de:
    - **Força Bruta:** Tentar adivinhar senhas de usuários usando dicionários.
    - **Pass-the-Hash:** Usar o hash da senha de um usuário, em vez da própria senha, para autenticar em outros sistemas na rede. Isso é uma tática comum em cenários de movimento lateral.
    - **Pass-the-Ticket:** Usar um ticket Kerberos comprometido (obtido através de outros ataques) para acessar serviços na rede.
- **Movimento Lateral:** Depois de obter credenciais, o CME pode ser usado para se mover por outras máquinas na rede. Ele pode executar comandos remotamente, criar sessões de shell, e até mesmo infectar outras máquinas com backdoors ou malware, tudo de forma automatizada.
- **Módulos e Scripts:** Uma das grandes vantagens do CME é a modularidade. Ele possui uma vasta biblioteca de módulos (chamados de "cme_modules") que permitem a execução de tarefas específicas, como:
    - Detectar vulnerabilidades conhecidas (como a vulnerabilidade "BlueKeep").
    - Filtrar arquivos de um servidor.
    - Desativar ferramentas de segurança (como o Windows Defender).
    - Verificar a política de senhas do domínio.

- **Geração de Relatórios:** Ele pode gerar relatórios detalhados das atividades de um teste de penetração, incluindo as credenciais que foram comprometidas, as vulnerabilidades encontradas e os hosts que foram acessados.

Embora o Hydra seja um poderoso atacante de força bruta para diversos protocolos, o CrackMapExec é uma ferramenta mais especializada e abrangente para *pentest* em domínios Windows. Enquanto o Hydra é como um "canhão" que ataca a autenticação de diversos serviços (SMB, FTP, SSH, etc.), o CME se concentra em explorar as vulnerabilidades específicas dos ambientes Windows e Active Directory, combinando enumeração, autenticação e movimento lateral em um único fluxo de trabalho.

2.4.2 Ferramentas de geração de palavras

No campo da segurança da informação, a geração de listas de senhas (*wordlists*) é uma etapa fundamental para ataques de força bruta e testes de penetração em ambientes controlados. Diversas ferramentas disponíveis em sistemas Linux permitem criar senhas a partir de uma senha base ou de padrões definidos.

Uma das ferramentas mais conhecidas é o Crunch, que possibilita a criação de *wordlists* especificando o conjunto de caracteres a ser utilizado, bem como o comprimento mínimo e máximo das senhas. Essa abordagem permite a construção de listas exaustivas ou personalizadas, o que torna a ferramenta bastante flexível (KALI, 2024).

Outra opção é o Custom Word List Generator (CeWL)**,** desenvolvido em Ruby, que extrai palavras de sites da web para compor dicionários de senhas. Esse tipo de recurso é útil em ataques direcionados, nos quais o atacante procura utilizar palavras relacionadas ao contexto da vítima, como nomes de produtos, termos técnicos ou jargões específicos (KALI, 2024).

O Common User Password Profiler (CUPP) segue uma lógica semelhante, mas parte de informações pessoais para gerar as combinações. Por exemplo, dados como nome, data de nascimento, apelidos e números relevantes podem ser utilizados para criar senhas prováveis, refletindo padrões comuns de escolha de credenciais por usuários (CYBRARY, 2025).

Além disso, ferramentas como o wordlister e o Psudohash foram criadas especificamente para modificar palavras-base. O Wordlister realiza combinações que incluem técnicas como *leetspeak* e capitalização, enquanto o Psudohash imita padrões de criação de senhas humanas, aplicando mutações realistas como inserção de números sequenciais ou símbolos comuns (GEEKSFORGEEKS, 2025; GITHUB, 2025).

Outra ferramenta bastante usada é o Pwgen, que gera senhas aleatórias de acordo com parâmetros como quantidade de caracteres, inclusão de símbolos e exclusão de caracteres ambíguos. Embora seja mais voltada para criação de senhas únicas e seguras, pode também ser utilizada para gerar listas automatizadas (OPENSOURCE, 2021).

Já o John the Ripper, é uma das ferramentas mais tradicionais de quebra de senhas, possui também funcionalidades para geração e mutação de listas. Ele permite aplicar regras de transformação sobre dicionários existentes, de modo a criar variações automáticas de palavras, como substituição de letras por números, adição de sufixos e alteração de capitalização. Isso o torna útil não apenas na quebra de senhas, mas também na criação de listas expandidas a partir de bases simples (OPENWALL, 2025). Por fim, comandos nativos do Linux, como aqueles que utilizam `/dev/urandom`, `tr` e `openssl rand`, também são frequentemente empregados para gerar strings aleatórias. Apesar de simples, essas abordagens permitem criar rapidamente listas de senhas de alta entropia diretamente pela linha de comando (UNIX STACKEXCHANGE, 2015).

2.5 Windows Server 2022

O Windows Server 2022 é uma versão da família de sistemas operacionais para servidores da Microsoft, lançada oficialmente em agosto de 2021. Ele foi projetado para fornecer alta disponibilidade, segurança aprimorada, desempenho otimizado e integração com ambientes em nuvem híbrida, por meio do Azure e outras tecnologias (MICROSOFT, 2021).

Voltado tanto para empresas de pequeno porte quanto para grandes corporações, o Windows Server 2022 traz melhorias significativas nos protocolos de rede, criptografia com a *Transport Layer Security* (TLS) 1.3, Segurança Baseada em Virtualização — Virtualization-Based Security (VBS) e suporte estendido a containers e virtualização (MACHADO; KALINOWSKI, 2022).

Uma das funcionalidades mais importantes desse sistema operacional é o Active Directory Domain Services (AD DS), conhecido simplesmente como Active Directory (AD), que é essencial para a gestão centralizada de usuários, dispositivos e políticas de segurança em redes corporativas.

2.5.1 Active Directory

O Active Directory é um serviço de diretório que permite organizar, gerenciar e autenticar recursos de uma rede, incluindo contas de usuários, grupos, estações de trabalho,

servidores, impressoras e políticas. Ele atua como o "cérebro da rede", garantindo que os usuários acessem os recursos certos com as permissões adequadas (BARROS, 2021).

No contexto de uma rede empresarial, o Active Directory permite:

- Autenticação centralizada: os usuários utilizam um único login para acessar recursos da rede (*Single Sign-On*);
- Gerenciamento de permissões: os administradores podem definir políticas de acesso baseadas em grupos, departamentos ou funções;
- Aplicação de políticas de segurança (*Group Policy Object* — GPOs): configurações como bloqueio de tela, restrição de dispositivos USB e instalação de programas, podem ser aplicadas remotamente;
- Escalabilidade: permite a criação de múltiplos domínios e árvores hierárquicas, facilitando a administração de grandes ambientes;
- Integração com o *Domain Name System* (DNS) e *Dynamic Host Configuration Protocol* (DHCP): trabalha em conjunto com outros serviços de rede para garantir que os dispositivos sejam corretamente identificados e localizados.

Além disso, com o Windows Server 2022, o AD ganha maior integração com o Azure Active Directory, permitindo uma gestão híbrida de usuários locais e em nuvem o que é uma tendência cada vez mais comum em organizações que utilizam ferramentas como Microsoft 365 (MICROSOFT, 2022).

A implementação do Active Directory em redes organizacionais permite reduzir falhas operacionais, aumentar o nível de segurança e conformidade com legislações como a LGPD, além de facilitar auditorias e o rastreamento de acessos indevidos.

O AD também permite o gerenciamento de grupos de acesso em um domínio, essa funcionalidade permite que usuários acessem pastas de arquivos na rede e dispositivos como impressoras. Todo processo que necessita de autorização e autenticação do usuário, passa pelo AD como uma requisição.

2.6 Ubuntu Server

O Ubuntu Server é uma distribuição Linux desenvolvida pela Canonical, voltada especificamente para ambientes de servidor. Robusto, gratuito e altamente personalizável, o sistema é amplamente utilizado em datacenters, serviços de nuvem e por profissionais de

segurança da informação para monitoramento de redes, análise de tráfego e resposta a incidentes (CANONICAL, 2022).

Graças à sua compatibilidade com uma ampla variedade de ferramentas *open source*, o Ubuntu Server é ideal para a implementação de infraestruturas de monitoramento e defesa cibernética, tanto em ambientes corporativos quanto acadêmicos ou experimentais.

O Ubuntu Server permite a instalação e configuração de diversas soluções de monitoramento, como:

- Nagios e Zabbix: para supervisão de disponibilidade e performance de hosts e serviços;
- Netdata e Prometheus + Grafana: para coleta de métricas e visualização em tempo real;
- Wireshark: para análise manual e detalhada de pacotes;
- Iptables e Uncomplicated Firewall (UFW): para firewall e controle de tráfego;
- Logwatch e Fail2ban: para análise de logs e bloqueio de IPs maliciosos;
- Suricata: inspeção de pacotes em tráfego.

A criação de ambientes controlados de laboratório para simular ataques cibernéticos e testar estratégias de defesa tornou-se uma prática fundamental no estudo da cibersegurança. Esses ambientes, também conhecidos como *testbeds* ou redes de laboratório, possibilitam a experimentação segura de técnicas ofensivas e defensivas, sem afetar sistemas reais. Eles são amplamente utilizados em treinamentos, pesquisas acadêmicas, auditorias e desenvolvimento de soluções de segurança (BARROS, 2021).

Entre os principais componentes de um ambiente de simulação realista, destacam-se: um sistema atacante (como o Kali Linux), um sistema alvo com serviços corporativos simulados (como o Windows Server 2022 com Active Directory) e um sistema de monitoramento e defesa (como o Ubuntu Server rodando o Suricata).

2.6.1 Monitoramento e Segurança com Suricata

O Ubuntu Server oferece suporte nativo à instalação e integração de ferramentas avançadas que permitem detectar falhas, analisar desempenho e identificar ameaças cibernéticas em tempo real.

Entre os destaques está o Suricata, uma poderosa ferramenta de código aberto que atua como *Intrusion Detection System* (IDS) e *Intrusion Prevention System* (IPS). Desenvolvido pela

Open Information Security Foundation (OISF), o Suricata analisa o tráfego de rede em tempo real, detectando padrões de ataque, acessos indevidos, malware e anomalias de comportamento (OISF, 2023).

Instalado em um servidor Ubuntu, o Suricata pode ser configurado para:

- Monitorar pacotes em tempo real (*deep packet inspection*);
- Detectar tentativas de força bruta, escaneamento de portas e *exploits* conhecidos;
- Bloquear automaticamente conexões suspeitas quando operando como IPS;
- Gerar logs e alertas integrados com outras ferramentas, como Elasticsearch Logstash Kibana (ELK) Stack ou Grafana.

A eficiência e flexibilidade fazem do Suricata uma solução viável até mesmo para redes domésticas com foco em segurança avançada, sendo possível rodá-lo em equipamentos leves, como mini-PCs ou servidores dedicados com Ubuntu (CERT.BR, 2023).

O poder do Suricata reside na sua capacidade de analisar o tráfego de rede de forma profunda e inteligente. As funcionalidades mais importantes incluem:

- **Inspeção Profunda de Pacotes (Deep Packet Inspection - DPI):** O Suricata não se limita a analisar os cabeçalhos dos pacotes (endereços IP e portas). Ele é capaz de inspecionar o conteúdo (*payload*) de cada pacote, procurando por padrões ou assinaturas que correspondam a ameaças conhecidas.

- **Análise Multicamada de Protocolos:** A ferramenta tem a capacidade nativa de captar diversos protocolos de aplicação, como HTTP, DNS, TLS e o SMB. Isto permite a criação de regras muito mais precisas que detectam anomalias específicas de cada protocolo.

- **Detecção Baseada em Assinaturas:** É o uso de regras (ou assinaturas). Ele compara o tráfego de rede com uma base de dados que contém milhares de assinaturas de ataques conhecidos, como tentativas de força bruta, scans de portas e exploração de vulnerabilidades.

- **Arquitetura Multi-Threaded:** Uma das maiores vantagens do Suricata sobre ferramentas mais antigas é que ele foi projetado desde o início para ser "multi-threaded". Isto significa que ele pode utilizar todos os núcleos de um processador

moderno para inspecionar o tráfego em paralelo, garantindo um alto desempenho mesmo em redes com grande volume de dados.

O Suricata pode ser implementado em duas arquiteturas distintas que definem a capacidade de resposta:

- **Modo IDS (Sistema de Detecção de Intrusão):** Neste modo, o Suricata é implementado "na lateral" da rede (*out-of-band*), onde recebe uma cópia do tráfego. A sua função é puramente passiva: ele **detecta e alerta**, mas não interfere no fluxo de tráfego.

- **Modo IPS (Sistema de Prevenção de Intrusão):** Neste modo, o Suricata é implementado "em linha" (*in-line*), atuando como uma ponte ou gateway, ele não só detecta as ameaças, mas também pode ser configurado para bloquear ativamente as conexões suspeitas, descartando os pacotes maliciosos antes que eles atinjam o alvo, também é possível responder ataques cibernéticos através da implementação de códigos e scripts customizados.

A capacidade de gerar logs detalhados é fundamental para qualquer ferramenta de segurança. O Suricata produz vários tipos de logs, sendo os mais importantes:

- **fast.log:** Um log de alertas de linha única, ideal para testes rápidos e para a integração com scripts simples.

- **eve.json:** O log de eventos unificado e moderno, utiliza o formato *JavaScript Object Notation* (JSON) que é estruturado e ideal para ser enviado e processado por ferramentas Security Information and Event Management (SIEM) como: Splunk, ELK Stack, etc. Uma única entrada neste log pode conter o alerta, detalhes do fluxo de rede e metadados do protocolo, oferecendo uma visão muito mais completa do incidente (OISF, 2023).

O Suricata detecta atividades maliciosas específicas, como tentativas de força bruta e scans de portas. Essa eficiência permite que seja utilizado numa grande variedade de cenários, desde grandes datacenters até redes menores que necessitem de segurança avançada, representando uma camada essencial para o monitoramento e defesa contra ameaças cibernéticas.

2.7 Firewall de Aplicação Web

O Firewall de Aplicação Web (WAF – *Web Application Firewall*) é uma solução de segurança especializada na proteção da camada de aplicação (Camada 7 do modelo OSI), com foco no monitoramento e filtragem de tráfego HTTP e HTTPS. Seu objetivo principal é proteger aplicações web contra ataques específicos, como Injeção de SQL (*SQL Injection*), Cross-Site Scripting (*XSS*), *Cross-Site Request Forgery* (CSRF) e outras ameaças direcionadas à lógica da aplicação (OWASP, 2023).

Diferentemente de um IPS de rede, como o Suricata utilizado neste projeto, o WAF atua analisando o conteúdo semântico das requisições web, compreendendo parâmetros, sessões e comportamento da aplicação. Já o IPS de rede concentra-se na inspeção de protocolos de infraestrutura e transporte, como SMB, TCP, UDP e ICMP, analisando padrões de tráfego e assinaturas de ataques em nível de rede (OISF, 2023).

No contexto deste trabalho, essa distinção é fundamental. A solução desenvolvida tem como foco a proteção da infraestrutura de autenticação interna, especialmente contra ataques de força bruta direcionados ao Active Directory via protocolo SMB. Em uma arquitetura corporativa real, um WAF estaria posicionado na DMZ, protegendo aplicações web expostas à Internet, enquanto o sistema IPS proposto atuaria na rede interna, protegendo os serviços de identidade e autenticação.

A atuação conjunta de um Firewall de Aplicação Web (WAF) e de um Sistema de Prevenção de Intrusões de rede (IPS) é considerada uma das abordagens mais eficazes para a proteção de ambientes corporativos modernos, pois cada tecnologia é especializada em camadas distintas do modelo OSI e em diferentes superfícies de ataque. Enquanto o WAF concentra-se na proteção das aplicações expostas à Internet, mitigando ataques direcionados à lógica de negócio e à camada de aplicação, o IPS de rede atua na identificação e bloqueio de comportamentos maliciosos em nível de infraestrutura, como varreduras de portas, tentativas de exploração de serviços e ataques de força bruta a protocolos de autenticação (OISF, 2023).

Essa complementaridade permite que ameaças sejam mitigadas em diferentes estágios do ataque, reduzindo significativamente a probabilidade de comprometimento total da infraestrutura. Em um cenário típico, ataques provenientes da Internet são inicialmente filtrados pelo WAF na DMZ, limitando o impacto sobre aplicações públicas. Caso um vetor de ataque consiga ultrapassar essa camada ou explorar outro ponto da rede, o IPS interno atua como uma segunda linha de defesa, protegendo serviços críticos e impedindo movimentos laterais dentro

da rede corporativa. Dessa forma, a combinação dessas tecnologias materializa o princípio da defesa em profundidade, no qual múltiplas camadas de segurança independentes trabalham de forma integrada para aumentar a resiliência do ambiente contra ataques cada vez mais sofisticados (OWASP, 2023; CERT.BR, 2023).

2.8 Software Livre de Licenciamento Open Source

O uso de soluções open source constitui um dos pilares fundamentais da arquitetura proposta neste trabalho, especialmente no que se refere ao baixo custo de implementação e à viabilidade para pequenas e médias empresas (PMEs). Para compreender esse aspecto, é essencial abordar os fundamentos do licenciamento de software livre, em especial a Licença Pública Geral GNU (GPL – *General Public License*), que rege tanto o sistema operacional Linux quanto o mecanismo de detecção de intrusões Suricata (GNU, 2007).

A licença GPL garante aos usuários quatro liberdades essenciais:

1. A liberdade de executar o programa para qualquer finalidade;
2. a liberdade de estudar como o programa funciona e adaptá-lo às próprias necessidades;
3. a liberdade de redistribuir cópias;
4. a liberdade de modificar o software e distribuir versões modificadas (GNU, 2007).

No contexto deste projeto, essas liberdades permitem que a solução de segurança seja implementada, customizada e mantida sem custos de licenciamento, diferentemente das soluções proprietárias, como firewalls de próxima geração da Cisco, Fortinet ou Palo Alto Networks, que dependem de modelos de assinatura anual para atualização de regras e funcionalidades avançadas (FORTINET, 2024; CISCO, 2024).

Dessa forma, o uso de software livre não apenas reduz barreiras financeiras, mas também amplia a autonomia técnica das organizações, tornando a solução desenvolvida neste trabalho especialmente adequada para PMEs que buscam elevar sua postura de segurança sem comprometer a sustentabilidade financeira do negócio.

3 Metodologia

3.1 Relatório do Incidente

O projeto foi desenvolvido a partir do relatório de um incidente de segurança ocorrido em uma organização. Inicialmente, um computador dentro do domínio foi comprometido por um software malicioso com capacidade de execução remota de código. A partir desse ponto, o atacante direcionou suas ações a um usuário específico dentro do domínio, executando um ataque por dicionário. Nesse tipo de ataque, o invasor tenta adivinhar a senha do alvo de forma exaustiva, utilizando referências conhecidas para gerar novas combinações e realizar centenas ou até milhares de tentativas de login.

O alvo pertencia a uma árvore do Active Directory recém-criada para gerenciar o acesso de terceiros à arquivos específicos dentro da rede corporativa. O primeiro sinal do incidente ocorreu quando a conta do usuário passou a ser bloqueada repetidamente, resultado da política de bloqueio após diversas tentativas de login sem sucesso. Inicialmente, os administradores consideraram hipóteses legítimas, como erro do próprio usuário ao digitar a senha ou até credenciais desatualizadas armazenadas em algum dispositivo conectado à rede. No entanto, a realidade era que o atacante estava testando os limites das políticas de segurança aplicadas e, assim, interrompendo o acesso legítimo do usuário.

Devido à demanda do usuário por acesso a arquivos e às interrupções causadas pelos bloqueios, a política de bloqueio de conta foi alterada para valores maiores de tentativas de login desse usuário. Essa decisão aumentou significativamente a superfície de ataque e expôs a conta e seus dados a um risco crítico. Durante esse período, o ataque de força bruta foi intensificado, resultando no comprometimento das credenciais do alvo.

O incidente só foi detectado após a análise detalhada dos arquivos de log, que revelaram centenas de tentativas de login em frações de segundo, evidenciando o uso de uma ferramenta maliciosa de automação de tarefa. As medidas cabíveis foram tomadas: o usuário alvo e o dispositivo de onde partiram os ataques foram identificados, os equipamentos foram formatados após coleta dos registros, e tanto senhas quanto nomes de usuário foram alterados. A partir desse evento, iniciou-se a investigação formal do ataque.

Apesar de o incidente envolver diferentes falhas de segurança como: intrusão de agente externo no domínio, comprometimento de host por código malicioso, execução remota de código, negação de serviço por bloqueio de contas e, posteriormente, ataque de força bruta. Este projeto foi direcionado ao estudo do ataque de força bruta para fins didáticos. A escolha seguiu

uma lógica de escalabilidade, partindo do ataque considerado “mais simples” até a reflexão sobre possíveis estratégias de prevenção contra ameaças mais complexas.

3.2 Ambiente de Estudo

A realização de experimentos em segurança da informação requer a utilização de ambientes de estudo controlados, pois ataques de rede, ainda que simulados, podem gerar impactos significativos se conduzidos em infraestruturas de produção. Esses ambientes, normalmente implementados por meio de máquinas virtuais, laboratórios isolados ou redes segmentadas, permitem que técnicas ofensivas, como varredura de portas, ataques de força bruta ou exploração de vulnerabilidades, sejam executadas sem comprometer sistemas reais. A principal importância de tais ambientes está na possibilidade de reproduzir cenários próximos aos encontrados em organizações, possibilitando a análise prática de ameaças e o entendimento de como diferentes tipos de ataque se propagam, ao mesmo tempo em que garante a segurança e a ética no processo de aprendizado.

Esses mesmos ambientes controlados são igualmente relevantes, uma vez que permitem a aplicação e o teste de mecanismos de proteção antes que sejam implantados em uma rede corporativa. Entre as principais formas de defesa que podem ser experimentadas estão a configuração de firewalls, a utilização de sistemas de detecção e prevenção de intrusões (IDS/IPS), a análise de tráfego por meio de ferramentas de monitoramento, bem como a aplicação de políticas de autenticação e controle de acesso mais robustas. Dessa forma, o estudo prático em ambientes isolados contribui não apenas para o desenvolvimento do conhecimento ofensivo, essencial ao profissional de cibersegurança, mas também fortalece a compreensão sobre estratégias defensivas que efetivamente reduzem os riscos de comprometimento em redes reais.

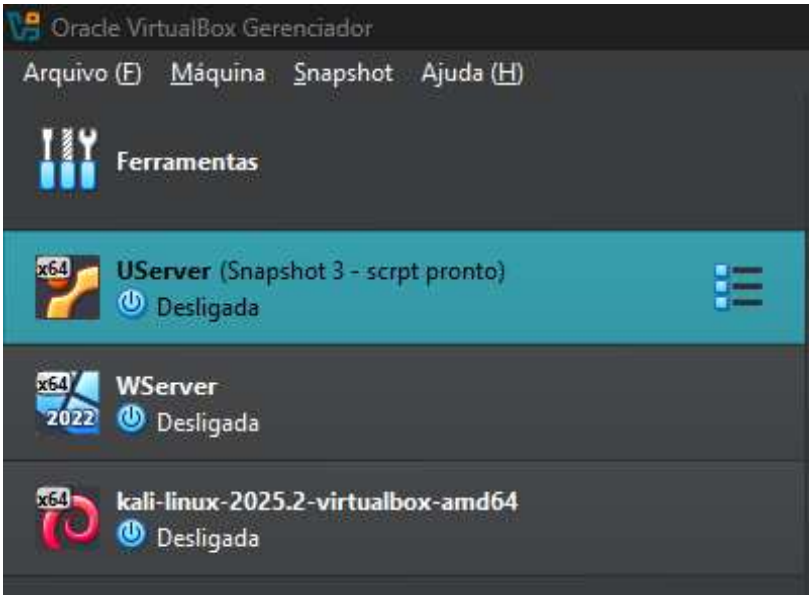
3.2.1 Virtual Box

O VirtualBox é um software de virtualização desenvolvido pela Oracle que permite criar e executar máquinas virtuais em um computador. Ele funciona como uma plataforma na qual é possível instalar e rodar diferentes sistemas operacionais de forma simultânea, sem a necessidade de instalar cada sistema diretamente no hardware físico. Isso significa que é possível ter, dentro de um mesmo computador, ambientes separados que simulam computadores independentes.

O VirtualBox aceita a instalação de uma grande variedade de sistemas convidados, além de emular diferentes tipos de rede, como *Network Address Translation* (NAT), bridge ou rede

interna. Também suporta múltiplos processadores, definição de memória RAM e configurações gráficas específicas para cada ambiente virtual (SERVERHUB BLOG, 2025).

Figura 1 – Interface do VirtualBox com máquinas virtuais instaladas



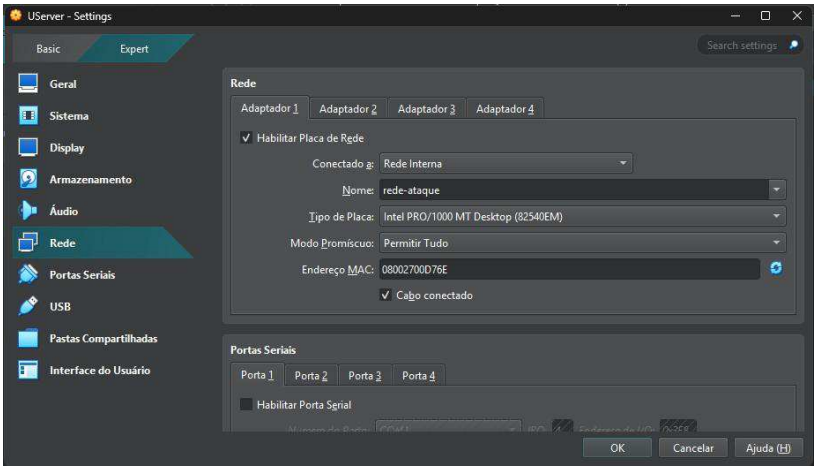
Fonte: AUTOR (2025)

Neste projeto, máquinas virtuais (VMs) do Ubuntu Server (UServer), Windows Server 2022 (WServer) e Kali Linux (kali-linux-2025.2-virtualbox-amd64) foram instaladas no VirtualBox como mostra a figura 1.

3.2.2 Configurações das Máquinas Virtuais

No VirtualBox é possível criar uma rede interna para simular a rede real onde o incidente ocorreu. Neste laboratório o Kali Linux representa o dispositivo que foi comprometido por software malicioso de execução de código remoto. A seguir, as configurações de rede de cada VM:

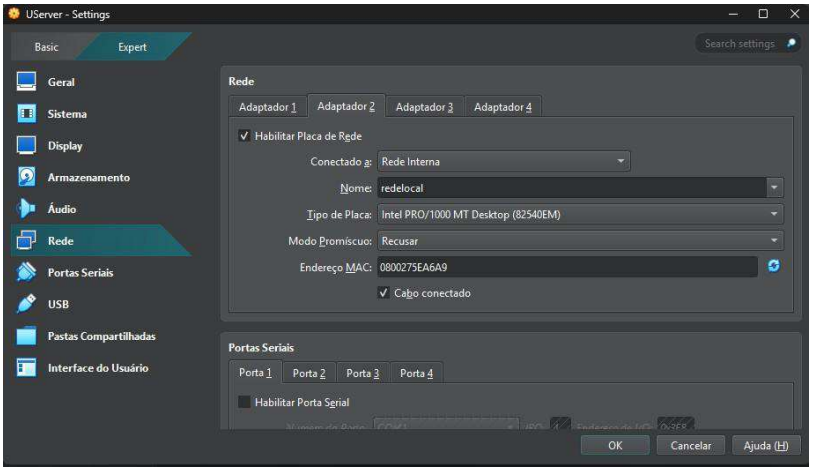
Figura 2 – Interface do adaptador de rede 1 da VM com Ubuntu Server no VirtualBox



Fonte: AUTOR (2025)

As configurações a serem observadas são: “Habilitar Placa de Rede” marcada para que a porta de rede esteja ativa quando inicializar a VM. A indicação de onde essa porta está conectada, nesse caso na “rede interna” do VirtualBox. O próximo campo é o nome da rede “rede-ataque” indicando uma sub-rede dentro da rede interna. O “Modo Promíscuo” selecionado para permitir tudo pois, essa porta específica precisa monitorar todo o tráfego de pacotes na sub-rede, não só os direcionados ao Ubuntu Server.

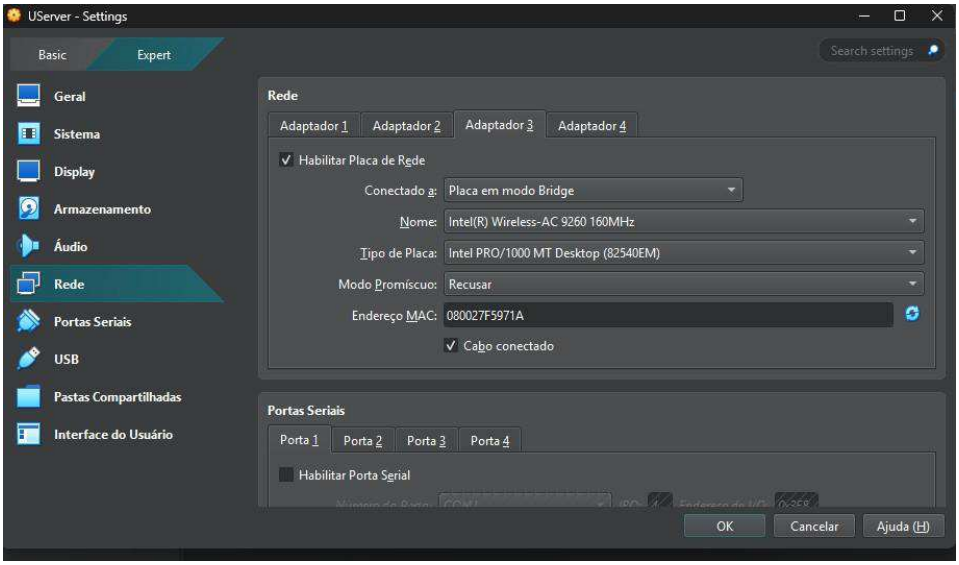
Figura 3 – Interface do adaptador de rede 2 da VM com Ubuntu Server no VirtualBox



Fonte: AUTOR (2025)

As configurações a serem observadas são: configuração da porta conectada à rede interna do VirtualBox, porém, em uma sub-rede diferente nomeada como “redelocal”. O modo promíscuo por usa vez está selecionado como “recusar” porque esta porta só precisa monitorar os pacotes direcionados a ela.

Figura 4 – Interface do adaptador de rede 3 da VM com Ubuntu Server no VirtualBox

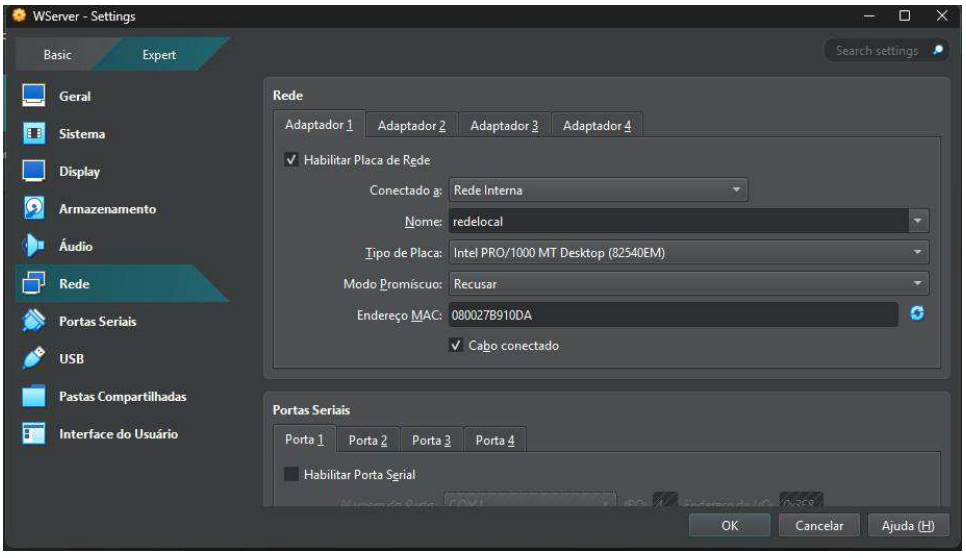


Fonte: AUTOR (2025)

As configurações a serem observadas são: configuração da porta como “Modo Brigde”, significa que esta porta está funcionando como uma ponte para a porta de rede da máquina hospedeira das VMs e dessa forma podendo se comunicar com outros dispositivos da rede física inclusive com a internet. O acesso à internet pelo Ubuntu Server garante a instalação de pacotes necessários e atualizações dos pacotes já instalados.

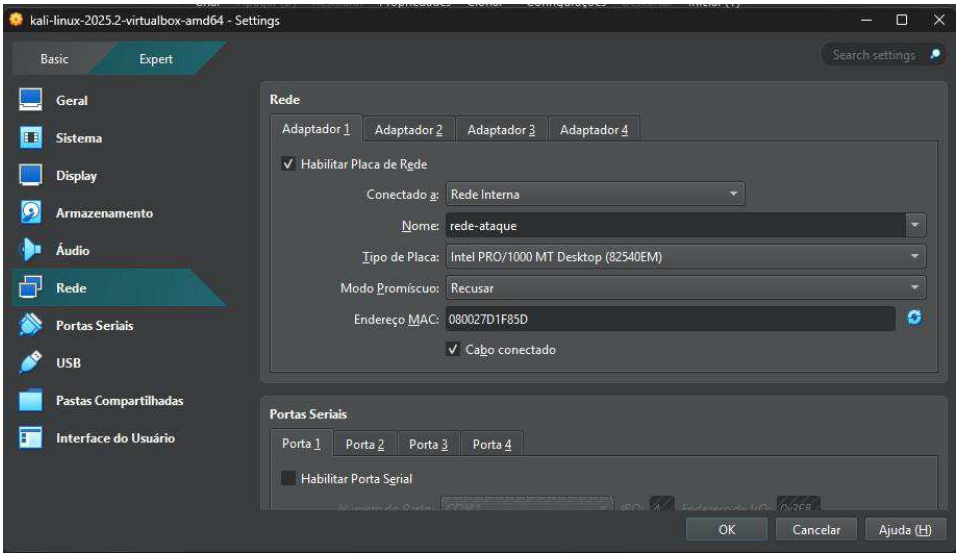
Para as VMs do Windows Server 2022 e Kali Linux apenas o adaptador de rede 1 foi configurado:

Figura 5 – Interface do adaptador de rede 1 da VM com Windows Server no VirtualBox



Fonte: AUTOR (2025)

Figura 6 – Interface do adaptador de rede 1 da VM com Kali Linux no VirtualBox



Fonte: AUTOR (2025)

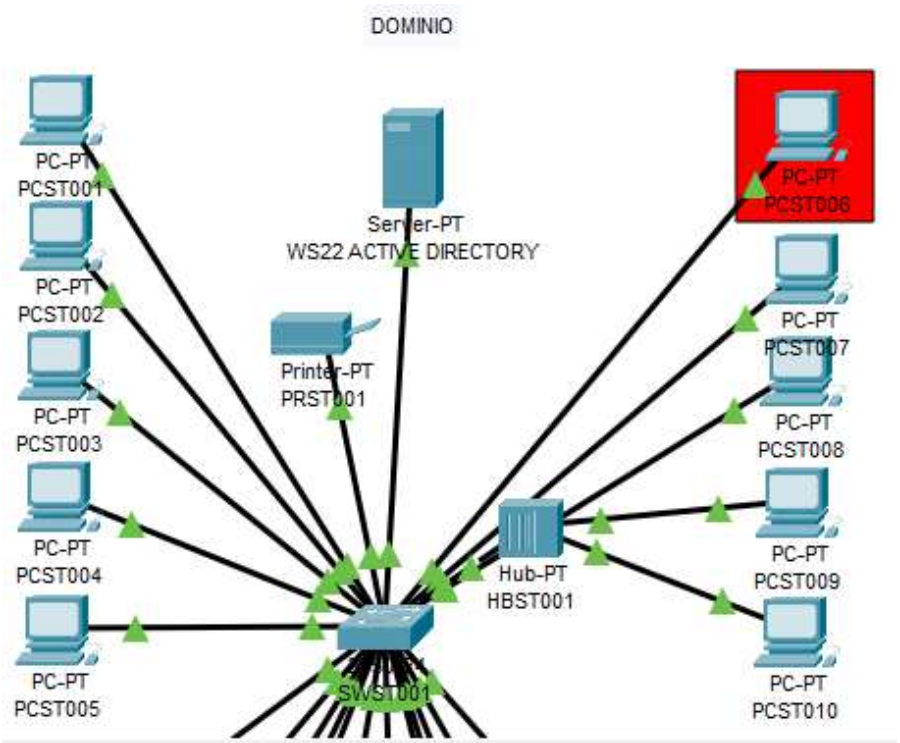
Cada uma destas duas VMs está conectada a uma sub-rede diferente (“redelocal” e “rede-ataque”) dentro da rede interna do VirtualBox. A explicação sobre essas duas sub-redes será mostrada nos próximos tópicos.

3.2.3 Disposição das Sub-redes no Domínio

A ausência de um sistema de monitoramento e proteção em uma rede corporativa expõe a infraestrutura a riscos significativos, uma vez que atividades maliciosas podem ocorrer sem serem detectadas. Sem mecanismos de detecção, como IDS ou IPS, ataques de força bruta, exploração de vulnerabilidades e movimentações laterais podem comprometer rapidamente serviços críticos, como o Active Directory, permitindo a escalada de privilégios e o controle indevido de recursos. Além disso, a falta de visibilidade dificulta a identificação de anomalias de tráfego e reduz a capacidade de resposta a incidentes, aumentando a probabilidade de indisponibilidade de serviços e vazamento de informações sensíveis.

Ao analisar o incidente ocorrido, é necessário partir do primeiro cenário onde a rede está sem monitoramento, no modelo da figura a seguir:

Figura 7 – Ilustração da rede não monitorada e desprotegida

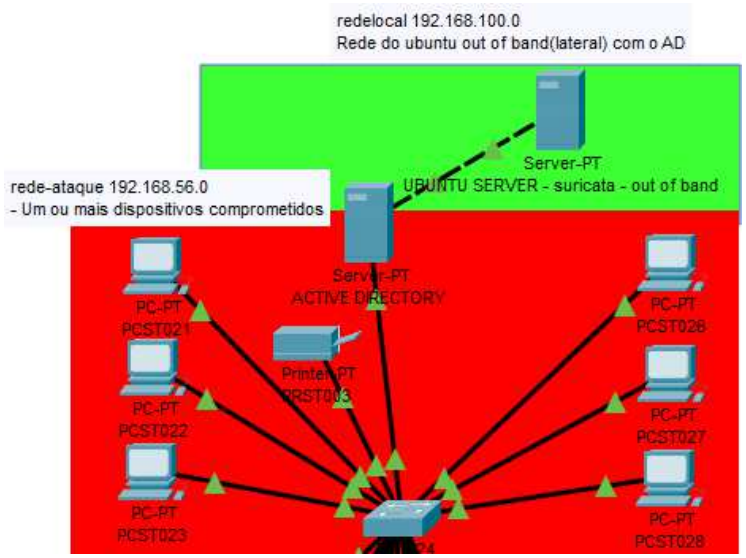


Fonte: AUTOR (2025)

Nesta rede ilustrativa um dos computadores conectados foi infectado por software malicioso como no incidente real. Conclui-se que a rede toda foi comprometida pois qualquer outro host poderia se tornar um alvo. Essa primeira falha de segurança deu início a uma série de explorações de vulnerabilidades na rede até o ponto crítico da execução de um ataque de força bruta.

A próxima configuração de rede é o segundo momento no estudo do incidente que parte da inserção do Ubuntu Server na topologia, para monitorar a porta de entrada de tráfego da rede do domínio no Windows Server.

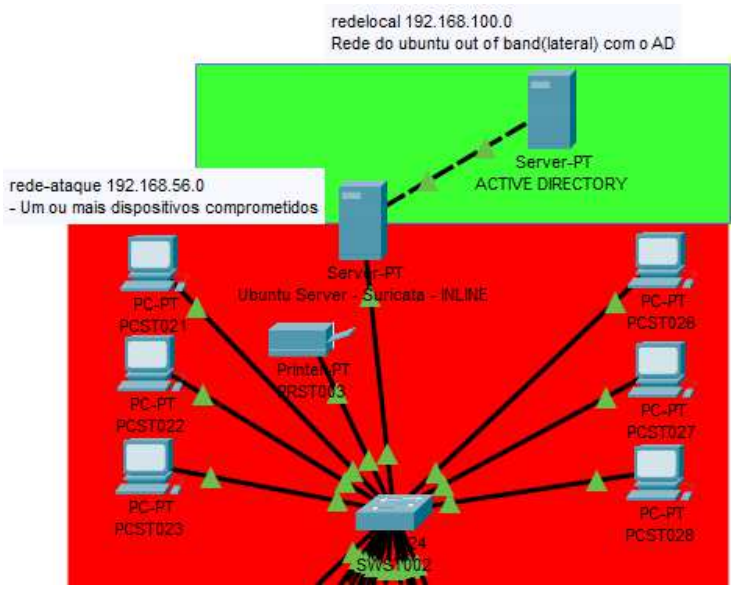
Figura 8 – Ilustração da rede monitorada com inserção do Ubuntu Server *Out-of-band*



Fonte: AUTOR (2025)

Nesta topologia o Ubuntu Server está na configuração *Out-of-band*, ou seja, na lateral do servidor do AD apenas “ouvindo” o tráfego que chega pela porta de rede. Existem duas sub-redes: a primeira configurada com o IP 192.168.56.0/24 e nomeada “rede-ataque”, é a rede do dispositivo de onde parte o ataque. A segunda configurada com o IP 192.168.100.0/24 e nomeada “redelocal”, é a rede entre o servidor do AD e o Ubuntu Server responsável pelo monitoramento.

Figura 9 – Ilustração da rede monitorada e protegida com inserção do Ubuntu Server *Inline*



Fonte: AUTOR (2025)

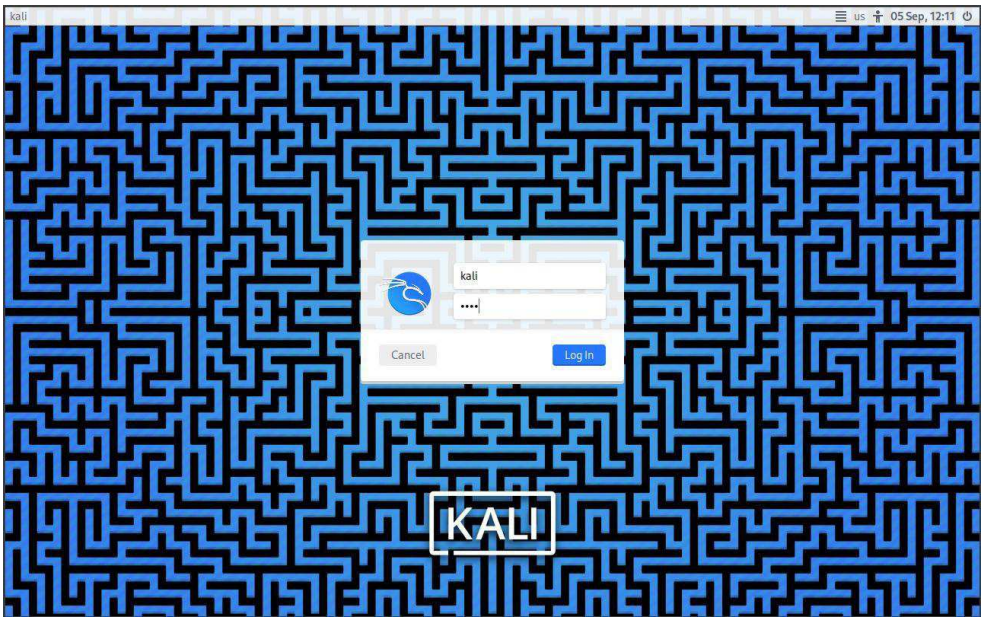
A principal diferença nesta última topologia é o Ubuntu Server colocado “a frente” do Server com AD. Isso muda completamente o cenário pois dessa forma todo o tráfego da rede que chega no AD será analisado pelo Suricata. As redes permanecem, “rede-ataque” identificada pelo IP 192.168.56.0/24 e a “redelocal” pelo IP 192.168.100.0/24 sendo a rede entre o Ubuntu Server e o servidor do AD. Com a demonstração das topologias de rede é possível ir mais a fundo na detecção e posteriormente proteção contra o ataque.

3.3 Ataque por Dicionário usando o Kali Linux

3.3.1 Predefinições de conectividade e parâmetros do ataque

Antes de mostrar como ataque é detectado e mitigado é importante focar no funcionamento. Este ataque é executado a partir do Kali Linux e tem como alvo um usuário presente numa rede corporativa gerenciada por um AD. Primeiro, efetua-se login no sistema operacional instalado em uma das VMs no VirtualBox.

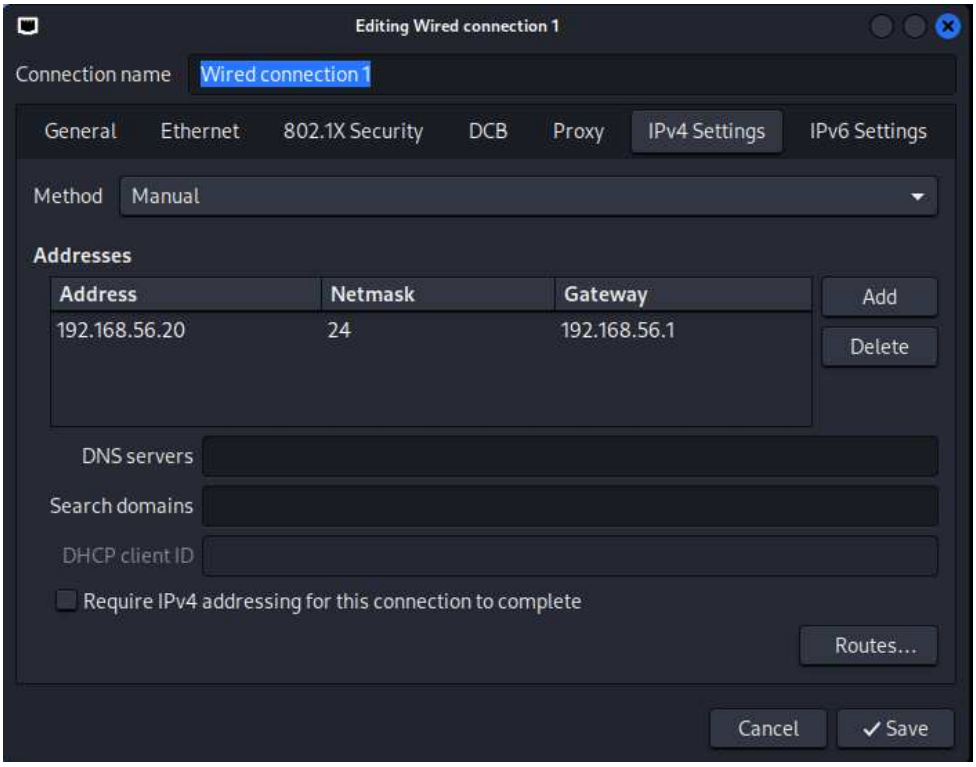
Figura 10 – Interface inicial do Kali Linux



Fonte: AUTOR (2025)

O Kali tem usuário definido como “kali” e senha “kali” por padrão, sendo possível alterar. Como foi mostrado na figura 6 a VM do Kali está conectada “fisicamente” à rede nomeada como “rede-ataque” e de IP 192.168.56.0/24. A partir daí, é necessário configurar uma porta de rede dentro do sistema.

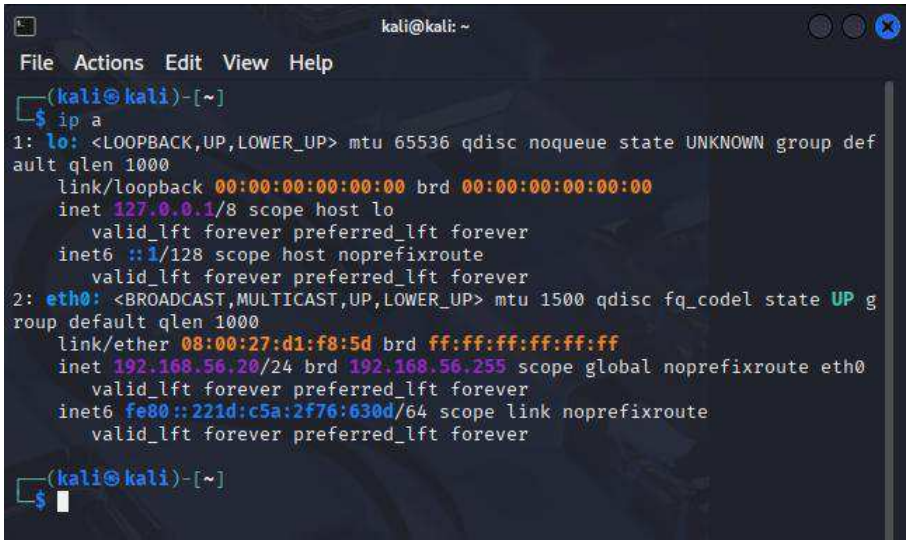
Figura 11 – Interface de configuração da porta de rede no Kali Linux



Fonte: AUTOR (2025)

Dentro do terminal do Kali executa-se o comando `ip a`. A seguir, a tela de configuração de rede.

Figura 12 – Interface de rede do Kali Linux após a configuração



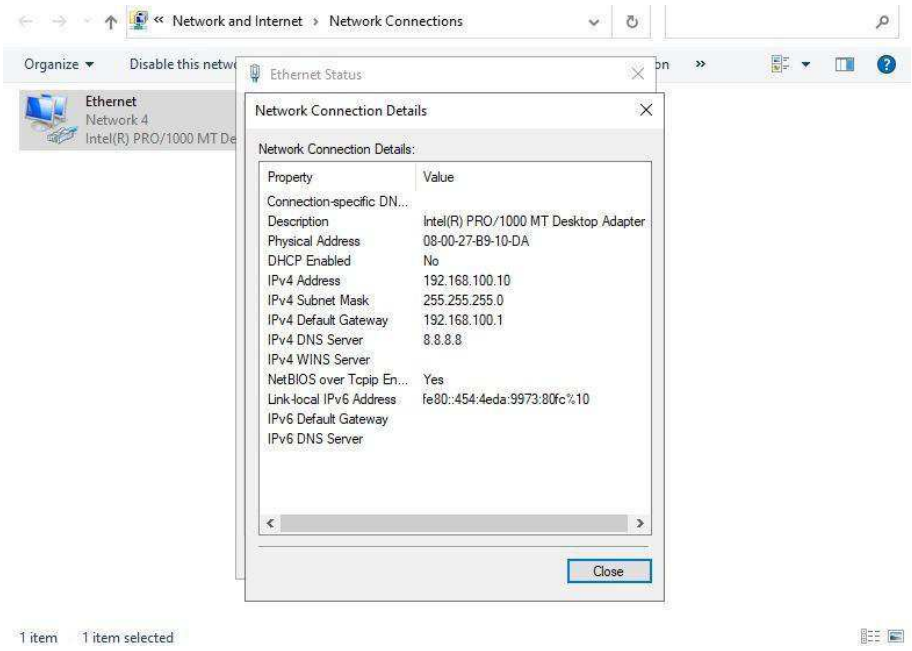
Fonte: AUTOR (2025)

Com essas duas imagens observa-se o seguinte: a porta de rede utilizada é a `eth0` configurada com o IP 192.168.56.20 que pertence a rede 192.168.56.0/24. No momento essas são as informações importantes para estabelecer a conexão com as demais VMs do projeto.

O Kali neste projeto representa o computador que foi infectado por software malicioso, sendo assim, em uma requisição de AD comum como por exemplo o login do usuário no host, a conexão partiria do host 192.168.56.20, segue para o gateway padrão dessa rede e chega no servidor do Active Directory onde a requisição é processada e retorna pelo mesmo grecaminho.

Quais informações sobre o alvo são necessárias para execução do ataque? De modo simplório, apenas do IP do alvo, nesse caso, o IP do Windows Server (na "redelocal") onde o AD está. A seguir, as informações de rede do Windows Server.

Figura 13 – Interface do adaptador de rede do Windows Server ligado à "redelocal".



Fonte: AUTOR (2025)

Observa-se que o IP do alvo é o 192.168.100.10 na rede 192.168.100.0/24.

Por que só o IP é o suficiente para a execução do ataque? Neste ataque utiliza-se a ferramenta de *pentest* presente no Kali Linux chamada CrackMapExec, especializada para ambientes Windows e AD. A estrutura do comando é formada pelo cabeçalho seguido do protocolo e os parâmetros de credenciais usados como mostra o quadro a seguir:

Quadro 1 – Comando de ataque por dicionário via Crackmapexec

```
crackmapexec smb 192.168.100.10 -u users.txt -p passwords.txt
```

Fonte: AUTOR (2025)

O `crackmapexec` é utilizado para automatizar a avaliação de segurança em grandes redes. Ele permite enumerar utilizadores, partilhas de rede, executar comandos remotamente e muito mais, caso encontre credenciais válidas.

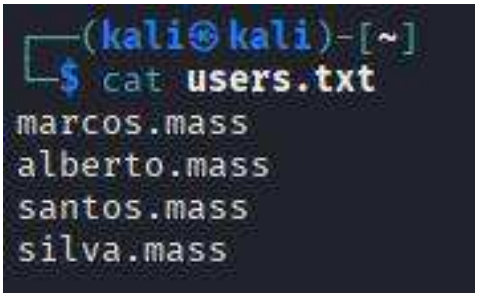
Já o `smb` especifica qual protocolo de rede será o alvo do ataque. O SMB refere-se ao Server Message Block, que é o protocolo principal utilizado pela Microsoft para compartilhamento de arquivos, impressoras e comunicação entre serviços em redes Windows.

Ao atacar o serviço SMB num Controlador de Domínio, como o Windows Server, cada tentativa de login é validada diretamente contra a base de dados do Active Directory. Por isso, é um alvo perfeito para verificar a validade de um par de utilizador e palavra-passe (usuário e senha).

Depois do servidor alvo de IP 192.168.100.10 e os argumentos `-u users.txt`, a flag "-u" indica que está sendo fornecido uma lista de utilizadores e "users.txt" é o nome do arquivo de texto que contém os potenciais nomes de usuário a serem testados, um por linha. Sabe-se que o alvo é um usuário e para preencher o primeiro arquivo "users.txt" com os possíveis nomes de usuário, a lógica é relativamente simples.

Em domínios Microsoft é muito comum o nome de usuário ser definido pelo nome real da pessoa que o utiliza, por exemplo, o nome Marcos Alberto Santos Silva, de acordo com o padrão normalmente utilizado, onde o usuário é definido pelo primeiro nome seguido por "ponto" e as iniciais do nome completo, assim, o primeiro nome de usuário disponível é: marcos.mass. Devido a política de credencias que impede que usuários sejam iguais, outra pessoa com o nome – por exemplo, Marcos Anderson Silva de Sousa, não poderia ter como usuário marcos.mass apesar de seguir o padrão citado. Ou seja, poderia ter como alternativas os nomes de usuário disponíveis: Anderson.mass, Silva.mass ou Sousa.mass. Por isso, ao obter o nome completo de um possível alvo, é simples supor qual seriam os nomes de usuário de acordo com esse padrão. Este projeto utilizou o nome fictício Marcos Alberto Santos Silva nos testes. Assim, o arquivo "users.txt" foi preenchido da seguinte forma:

Figura 14 – Conteúdo do arquivo “users.txt”.



Fonte: AUTOR (2025)

Já para o segundo argumento `-p passwords.txt`, a flag “-p” indica que está fornecendo uma lista de palavras-passe e “passwords.txt” é o nome do arquivo de texto que contém as senhas que serão testadas, uma por linha. Para preencher este arquivo é necessário o auxílio de ferramentas de geração de palavras a partir de uma ou mais palavras base com prefixos, sufixos e demais complementos. Neste projeto utilizou-se o John The Ripper dentro do Kali Linux para gerar tais palavras.

3.3.2 John The Ripper

O John the Ripper (JtR) tem um recurso chamado *wordlist mangling* que permite criar variações a partir de uma palavra raiz ou de um dicionário inicial, aplicando regras para gerar mais combinações (adicionar números, trocar letras por símbolos, inverter, etc.).

Primeiro um arquivo com a palavra base é criado.

Quadro 2 – Comando de criação do arquivo “basepass.txt” com a senha base

```
echo “Senha@atual2025” > basepass.txt
```

Fonte: AUTOR (2025)

Após a criação desse arquivo, executa-se o comando `jhon` com os parâmetros para geração de novas palavras-passe derivadas.

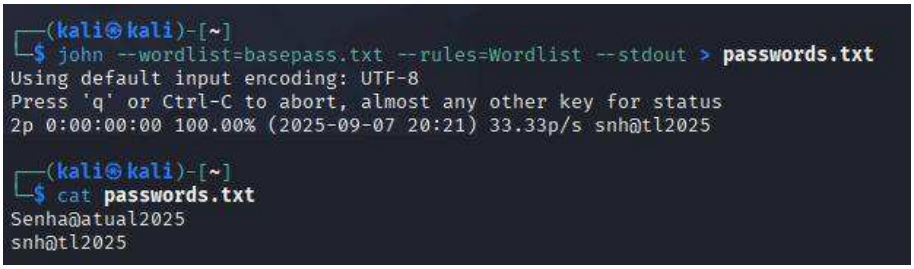
Quadro 3 – Comando de geração de senhas utilizando o John The Ripper

```
john –wordlist=basepass.txt –rules=Wordlist --stdout > passwords.txt
```

Fonte: AUTOR (2025)

Gerando a saída a seguir:

Figura 15 – Tela de saída do comando “john”



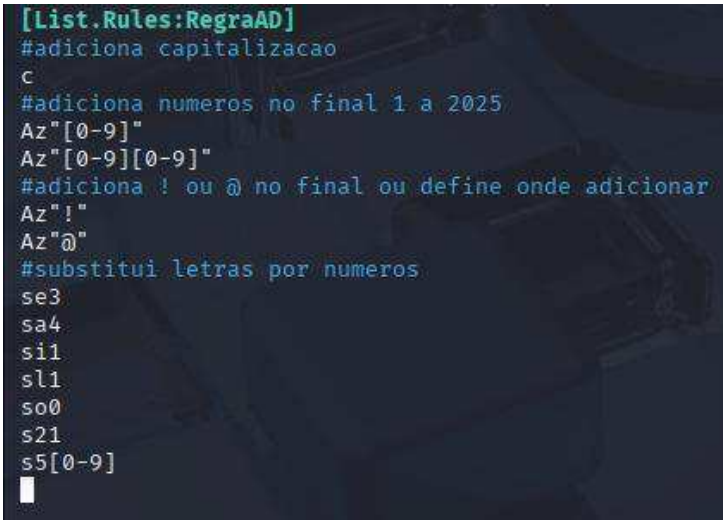
Fonte: AUTOR (2025)

Sobre os parâmetros deste comando: o `--wordlist=basepass.txt` é a indicação do arquivo de referência, `--rules = Wordlist` é a regra utilizada, “Wordlist” é uma regra padrão na qual as vogais são removidas e todo o resto da senha é mantido como é possível observar no fim da figura 15. Por último `--stdout > passwords.txt` é a indicação do arquivo destino onde as senhas geradas serão armazenadas. O ponto mais importante nesse comando é a regra que será aplicada. O exemplo utilizou a regra “Wordlist”, mas existem muitas outras regras prontas que podem ser usadas e também é possível utilizar todas, porém, o arquivo destino ficaria muito grande em armazenamento e quantidade de senhas geradas (na ordem de centenas de milhões).

Dado o conhecimento prévio do autor do ataque sobre padrões de senha em domínios Microsoft e até mesmo referencias de senhas antigas utilizadas pelo alvo em sites desprotegidos, cache de páginas vulneráveis e que foram exploradas, o ideal para este caso é utilizar uma regra personalizada para reduzir a quantidade de senhas geradas e consequentemente, também diminuindo a quantidade de tentativas de login executadas no ataque, já que para cada usuário listado no arquivo “users.txt”, todas as senhas no arquivo “passwords.txt” serão testadas.

A regra personalizada deixa todo o ataque mais eficiente pois, há uma probabilidade maior de a senha correta se encontrar no arquivo de senhas geradas. A regra personalizada foi criada no final do arquivo de parâmetros do John The Ripper instalado no Kali Linux no caminho /etc/john/john.conf.

Figura 16 – Conteúdo da regra personalizada “RegraAD”



Fonte: AUTOR (2025)

A regra criada “RegraAD” segue o padrão de uma ou mais senhas do conhecimento do autor do ataque que foram colocadas no arquivo “basepass.txt”. Ao executar novamente o comando `john` porém, com a nova regra personalizada como parâmetro, obtém-se o seguinte resultado:

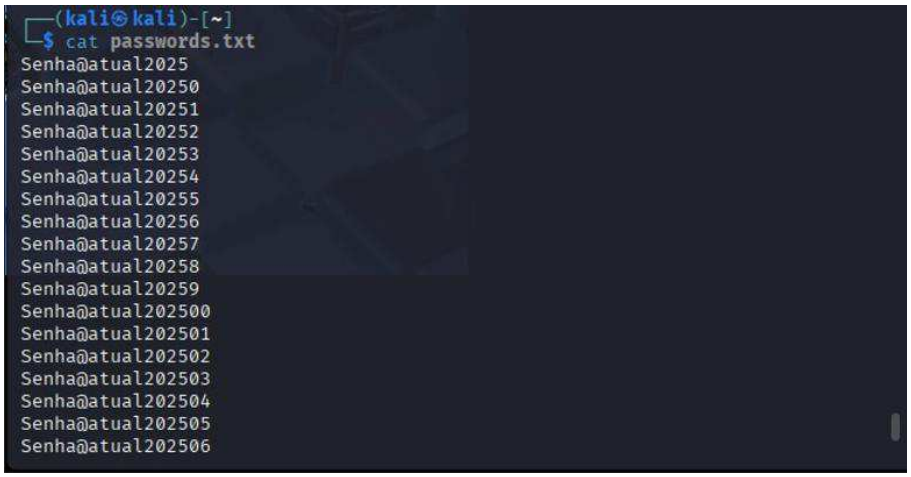
Quadro 4 – Comando de geração de palavras-passe integrado com regra personalizada

```
john –wordlist=basepass.txt –rules=RegraAD --stdout > passwords.txt
```

Fonte:AUTOR (2025)

Gerando a saída a seguir:

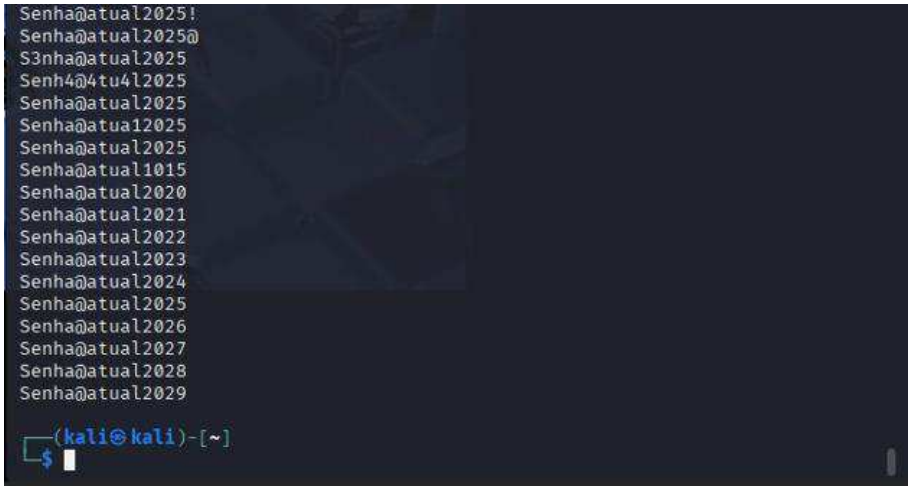
Figura 17 – Saída do comando “jhon” utilizando a regra personalizada “RegraAD”



Fonte:AUTOR (2025)

Mais de 300 senhas foram geradas e as primeiras tem apenas alguns dígitos adicionados no final, mas durante o arquivo outras partes da senha raiz foram alteradas seguindo o padrão estabelecido pela regra personalizada como mostra a figura a seguir:

Figura 18 – Últimas senhas geradas no arquivo “passwords.txt”



Fonte: AUTOR (2025)

Em algumas das últimas palavras geradas as letras foram substituídas por números de acordo com o comando presente na regra personalizada e observa-se que no arquivo também está presente a própria senha usada como base. Outro ponto importante é que quem está atacando pode selecionar senhas potenciais e excluir outras, com o intuito de reduzir a quantidade de tentativas de login realizadas.

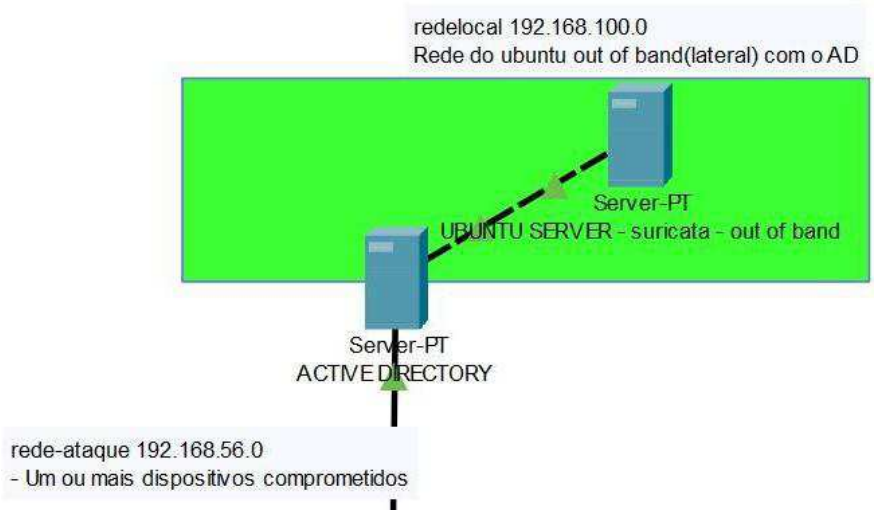
3.4 Sistema de Detecção de Intrusões

3.4.1 Topologia *Out-of-Band*

Um Sistema de Detecção de Intrusões é uma ferramenta que monitora uma rede e emite alertas ao detectar anormalidades como várias tentativas de acesso malsucedidas. Neste projeto toda parte de mitigação começa pela detecção de tais anormalidades.

Para implementar um IDS é necessário conectar o servidor do AD a um dispositivo com o Ubuntu Server instalado como sistema operacional e dessa forma, através do Suricata, é possível visualizar todo o tráfego de uma rede especificada nos arquivos de configuração do Suricata. Tal topologia está disposta da seguinte maneira:

Figura 19 – Topologia *Out-of-Band* para implementação de um IDS



Fonte: AUTOR (2025)

Esta imagem mostra a configuração *Out-of-Band*, onde o Suricata é configurado para monitorar o tráfego de uma rede.

3.4.2 Arquivo de configurações do Suricata

Todas as definições e parâmetros que o suricata precisa para monitorar uma rede ou desempenhar qualquer outra função estão presentes no arquivo "suricata.yaml" no caminho "/etc/suricata/suricata.yaml" dentro do Ubuntu Server.

Figura 20 – Definição de parâmetros para o monitoramento da rede

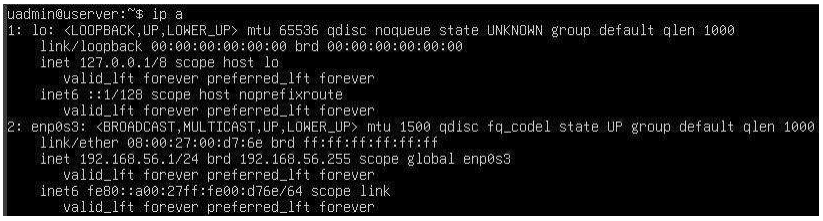
```
vars:
  # more specific is better for alert accuracy and performance
  address-groups:
    HOME_NET: "[192.168.56.0/24]"
    #HOME_NET: "[192.168.0.0/16]"
    #HOME_NET: "[10.0.0.0/8]"
    #HOME_NET: "[172.16.0.0/12]"
    #HOME_NET: "any"

    EXTERNAL_NET: "!$HOME_NET"
    #EXTERNAL_NET: "any"
```

Fonte: AUTOR (2025)

O primeiro ponto importante é definir qual rede será monitorada, como mostra a imagem, a rede 192.168.56.0/24 anteriormente nomeada "rede-ataque" foi incluída no parâmetro "HOME_NET". Esse parâmetro está relacionado às redes que o Ubuntu Server está conectado, de maneira lógica, só é possível monitorar rede que o servidor está conectado como mostra a imagem a seguir:

Figura 21 – Definições de rede do Ubuntu Server



Fonte: AUTOR (2025)

Ao inserir o comando `ip a` no terminal do Ubuntu Server observa-se que existe a interface padrão de “loopback” e logo depois a interface “enp0s3” com o IP 192.168.56.1 pertencente a mesma rede que será monitorada. Essa interface precisa ser inserida em outra parte do arquivo principal de configurações do Suricata.

Figura 22 – Definição da interface de rede a ser monitorada (arquivo “suricata.yaml”)

Fonte: AUTOR (2025)

Neste momento, o Suricata já consegue monitorar a rede, porém com regras padrões, então criou-se uma regra customizada focada em detectar ataques de força bruta pelo protocolo SMB.

3.4.3 Regra Customizada de Detecção de Ataques de Força Bruta

Regras customizadas são importantes para direcionar o sistema de detecção para um ou mais tipos de ataques. Em um ataque por dicionário utilizando o protocolo SMB para validar a autenticação de credenciais, uma regra customizada é ideal para detectá-lo.

Um arquivo nomeado “regra_custom.rules” foi criado com conteúdo:

Quadro 5 – Conteúdo da regra personalizada (arquivo “regra_custom.rules”)

```
alert tcp any any -> $HOME_NET 445 (msg:"smb atk id 5214"; sid:1000001;
rev:1;).
```

Fonte: AUTOR (2025)

O cabeçalho define a ação, o protocolo, a origem e o destino do tráfego a ser inspecionado e analisando cada parte:

- `alert`: esta é a ação a ser tomada quando a regra corresponde, “alert” significa "gere um alerta".

- `tcp`: este é o protocolo de rede que a regra irá inspecionar. O foco é o tráfego pelo *Transmission Control Protocol* (TCP), que é o protocolo usado pelo SMB. Outras opções seriam *User Datagram Protocol* (UDP) e *Internet Control Message Protocol* (ICMP), etc.

- `any any`: esta é a origem do tráfego, composta pelo IP de origem e porta de origem. A palavra-chave “any” significa "qualquer um". Portanto, esta regra irá aplicar-se a tráfego vindo de qualquer endereço IP e de qualquer porta de origem.

- `->`: este é o operador de direção, a seta indica que o tráfego está a fluir da origem (à esquerda) para o destino (à direita).

- `$HOME_NET 445`: este é o destino do tráfego, composto pelo IP de destino e porta 445, padrão para o serviço SMB (Server Message Block), que é o alvo do ataque no Active Directory.

Traduzindo o cabeçalho da regra: "Gere um alerta para qualquer tráfego TCP, vindo de qualquer lugar, que seja destinado a qualquer computador na minha rede local na porta 445."

No restante do ‘corpo’ da regra, as opções dentro dos parênteses separadas por ponto e vírgula fornecem os metadados e os detalhes da detecção:

- `msg: smb atk id 5214`, a palavra-chave “msg” define a mensagem de alerta. Esta é uma mensagem customizada que é usada para identificar o tipo de ataque e normalmente contém um “id” arbitrário para que outras ferramentas ou até mesmo para que um administrador da rede consiga identificar os alertas gerados a partir dessa regra.

- `sid:1000001`, a palavra-chave “sid” define o ID da assinatura (Signature ID). Este é um número único que identifica esta regra específica. A convenção é que regras personalizadas usem SIDs a partir de 1.000.000 para não entrarem em conflito com as regras padrão.

- `rev:1`, define o número da revisão. É usado para controle de versões. Se a regra fosse modificada no futuro, seria alterado para “rev:2”.

Agora a regra customizada precisa ser referenciada dentro do arquivo “suricata.yaml”.

Figura 23 – Inserção da regra customizada no arquivo de configurações do Suricata

Fonte: AUTOR (2025)

Na imagem, o caminho do arquivo de regras padrão e a referência do arquivo da regra customizada "/etc/suricata/regra_custom.rules". Agora a configuração dos parâmetros de monitoramento está pronta.

3.5 Sistema de Prevenção de Intrusões

3.5.1 Topologia *In-line*

Anteriormente o Suricata apenas alertava em mensagens de log, mas agora será capaz de interromper a execução de ataques contra o servidor do Active Directory.

A mudança da topologia de *out-of-band* ("na lateral") para *in-line* ("no caminho") é o passo crucial que transforma o Ubuntu Server de um Sistema de Detecção de Intrusão (IDS) passivo para um Sistema de Prevenção de Intrusão (IPS) ativo. A eficiência do bloqueio no modelo *in-line* é superior porque a ação é preventiva, e não apenas reativa.

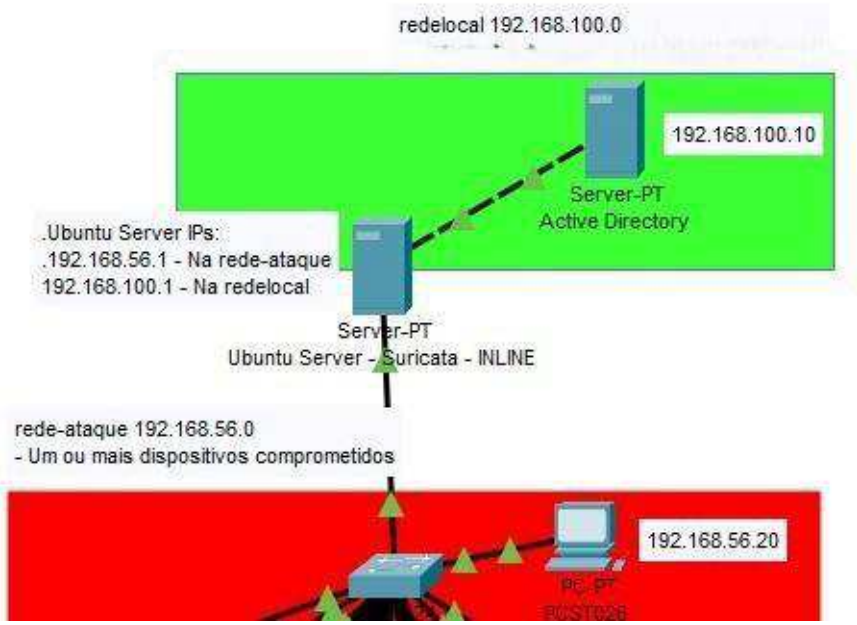
Na arquitetura inicial, o Ubuntu Server recebia apenas uma cópia do tráfego entre o atacante e o alvo. O fluxo original do ataque continuava o caminho sem interrupções até ao Windows Server. O Suricata analisava a cópia do pacote e, ao detectar a ameaça, gera um alerta. No entanto, neste ponto, o pacote malicioso original já foi entregue ao Windows Server.

Para bloquear o ataque, o sistema teria de executar uma ação reativa, como enviar um pacote TCP *Reset* para tentar fechar a conexão. Este método tem duas desvantagens significativas:

1. Atraso: O bloqueio só acontece "depois" de um ou mais pacotes do ataque já terem atingido o alvo.
2. Confiabilidade: A ação de resposta pode falhar ou ser ignorada pelo alvo, não garantindo a interrupção do ataque.

Devido a isso, foi criada uma topologia da rede:

Figura 24 – Topologia *in-line* para implementação de um IPS



Fonte: AUTOR (2025)

Ao posicionar o Ubuntu Server como um gateway, todo o tráfego é forçado a passar por ele para chegar ao Windows Server. Isto muda fundamentalmente a sua capacidade de defesa. O Suricata inspeciona o pacote de ataque original em tempo real, enquanto ele está em trânsito. Ao detectar o pacote como malicioso, o Suricata, operando em modo IPS, pode tomar a ação imediata de acionar recursos para descartar (*drop*) o pacote ou outros mecanismos de defesa.

Uma nova configuração deve ser feita no Ubuntu Server pois ao ser colocado na posição de gateway, é necessário configurar o roteamento de pacotes para que os demais dispositivos na rede tenham acesso ao servidor do AD.

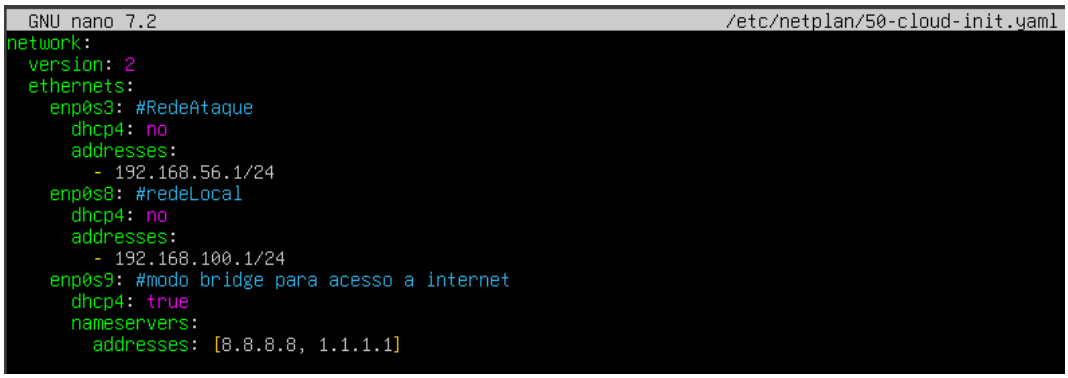
3.5.2 Configurações de Rede e Roteamento do Ubuntu Server

Para que o Ubuntu Server funcione como um roteador entre a “rede-ataque” e a “redelocal”, é necessário reconfigurar a interface de rede “enp0s3” e habilitar duas novas interfaces.

No VirtualBox é possível habilitar novas interfaces acessando as configurações de adaptadores de rede da VM como mostra a figura 2. Na VM do Ubuntu Server o adaptador de rede 1 foi reconfigurado para a “rede-ataque”, o adaptador de rede 2 configurado na “redelocal” e o terceiro adaptador foi configurado no “Modo Brigde” para conectar-se com a placa de rede da máquina hospedeira do VirtualBox e dessa forma, acessar a internet para viabilizar a instalação de novos pacotes no Ubuntu Server ou atualizações na VM do Windows Server 2022.

Dentro do Ubuntu Server as configurações de IP das interfaces precisam ser inseridas no arquivo de caminho “/etc/netplan/50-cloud-init.yaml” (nome padrão em muitas versões desse sistema operacional). Acessando este arquivo no Ubuntu Server, criam-se as seguintes definições:

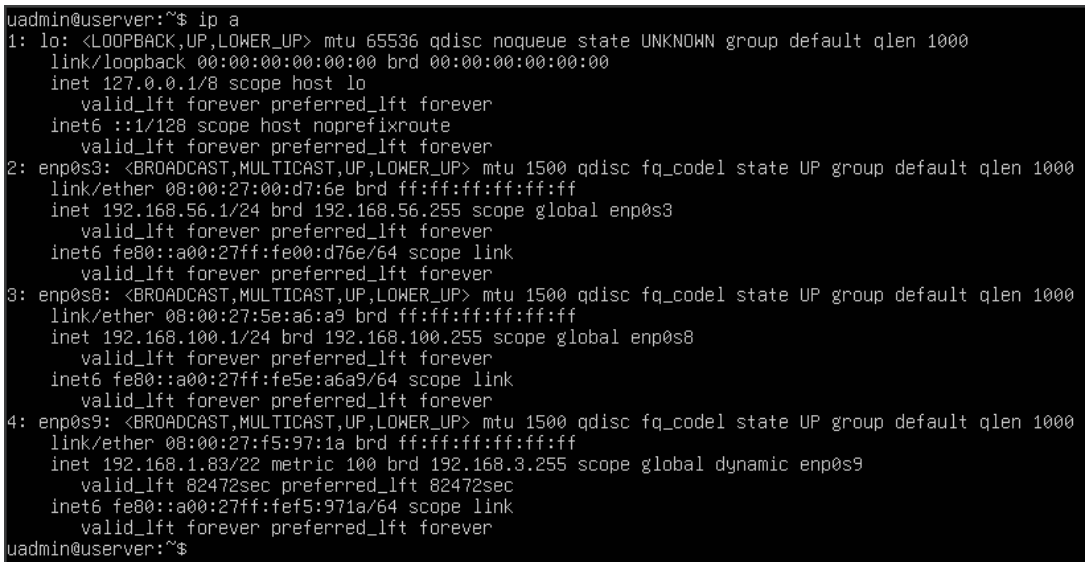
Figura 25 – Configurações das interfaces de rede (arquivo “50-cloud-init.yaml”)



Fonte: AUTOR (2025)

Posteriormente, a execução do comando `sudo netplan apply` aplica as definições e com o comando `ip a`, gerando a tela de saída a seguir:

Figura 26 – Interfaces de rede do Ubuntu Server



Fonte: AUTOR (2025)

Na imagem, a interface “enp0s3” está listada com o IP fixo 192.168.56.1 pertencente a “rede-ataque” (rede em que o host atacante também se encontra), a interface “enp0s8” com o IP fixo 192.168.100.1 pertencente a “redelocal” (rede entre o Ubuntu Server e o Windows

Server) e a interface “enp0s9” com o IP 192.168.1.83, este que foi atribuído via DHCP pois essa interface está em “modo bridge”. Outro ponto importante para a conexão é que o IP da interface “enp0s8” foi “setado” como gateway padrão no Windows Server, assim como a interface “enp0s3” se tornou a “ponte de conexão” entre a “rede-ataque” e a “redelocal”. Dessa forma, para requisições de AD chegarem ao servidor, é necessário passar primeiro pelo Ubuntu Server.

Um detalhe importante para a monitoramento efetivo das duas redes nesta nova topologia, é a adição da rede 192.168.100.0/24 (“redelocal”) no campo “HOME_NET” dentro do arquivo de configurações do Suricata (“suricata.yaml”) mostrado na figura 21.

Definir os IPs das interfaces não é o suficiente para garantir a que os pacotes passem de uma rede para outra, então é necessário configurar o roteamento de pacotes no Ubuntu Server.

Duas configurações principais no nível do sistema operacional foram necessárias: Habilitar o encaminhamento de IP e Configurar a tradução de endereços de Rede.

- Habilitar o Encaminhamento de IP (*IP Forwarding*): por padrão, um sistema Linux não encaminha pacotes entre as suas diferentes interfaces de rede. Esta funcionalidade precisa ser explicitamente ativada para que ele possa atuar como um roteador.
- Comando para Ativação Imediata**:** este comando altera o parâmetro do *kernel* em tempo real, ativando o roteamento instantaneamente.

Quadro 6 – Comando de encaminhamento de pacotes

```
sudo sysctl -w net.ipv4.ip_forward=1
```

Fonte: AUTOR (2025)

O comando `sysctl -w` escreve o valor 1 (que significa "ativado") no parâmetro do *kernel* `net.ipv4.ip_forward`.

Para garantir que o encaminhamento de IP configurado no quadro 6, permaneça ativo após uma reinicialização, a configuração foi salva num arquivo de sistema.

Quadro 7 – Comando de fixação de configurações

```
echo "net.ipv4.ip_forward=1" | sudo tee /etc/sysctl.d/99-forwarding.conf
```

Fonte: AUTOR (2025)

Este comando cria um arquivo de configuração dentro de “/etc/sysctl.d/” que é lido automaticamente durante a inicialização do sistema, aplicando a configuração `net.ipv4.ip_forward=1` de forma permanente.

Na configuração da tradução de endereços de rede, a VM do Windows Server (192.168.100.10) está numa rede privada e precisa de uma forma de aceder à internet. O NAT permite que o tráfego do Windows Server "pegue emprestado" o endereço IP da interface de internet do Ubuntu para comunicar com a internet. O comando a seguir cria a regra de NAT:

Quadro 8 – Comando de configurações do NAT

```
sudo iptables -t nat -A POSTROUTING -o enp0s9 -j MASQUERADE
```

Fonte: AUTOR (2025)

Explicação:

- `-t nat`: Especifica que é necessário modificar na tabela de NAT.
- `-A POSTROUTING`: Adiciona a regra à cadeia “POSTROUTING”, que processa os pacotes logo antes de eles saírem do servidor.
- `-o enp0s9`: A regra aplica-se a pacotes que estão a sair pela interface enp0s9.
- `-j MASQUERADE`: Esta é a ação. "Mascarar" significa trocar o endereço IP de origem do pacote (ex: 192.168.100.10) pelo endereço IP da interface de saída (enp0s9).

Para tornar esta regra permanente, foi utilizado o pacote *iptables-persistent*:

Quadro 9 – Comando de fixação de configurações do NAT

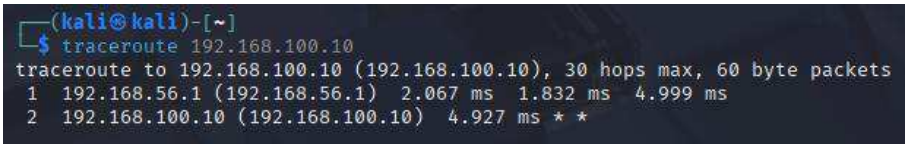
```
sudo netfilter-persistent save
```

Fonte: AUTOR (2025)

Este comando salva todas as regras *iptables* ativas em arquivos de configuração (“/etc/iptables/rules.v4”) que são lidos e reaplicados automaticamente a cada inicialização do sistema. Juntos, estes dois conjuntos de comandos transformam um Ubuntu Server padrão num gateway de rede funcional, capaz de encaminhar e traduzir o tráfego entre diferentes redes, estabelecendo a base para a implementação do IPS *in-line*.

Para testar a conexão entre as VMs executam-se os comandos de rastreamento de rota em cada uma. A saída no terminal do Kali Linux:

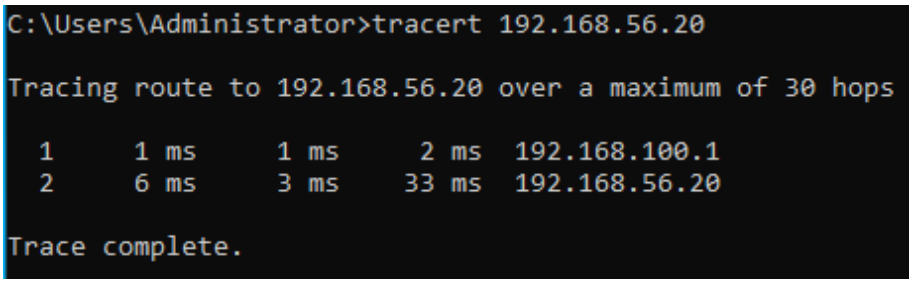
Figura 27 – Tela de rastreamento de rota do Kali Linux para o Windows Server



Fonte: AUTOR (2025)

No terminal do Windows Server:

Figura 28 – Tela de rastreamento de rota do Windows Server para o Kali Linux



Fonte: AUTOR (2025)

3.5.3 Implementação do Script de Resposta Automática

A implementação de um Sistema de Detecção de Intrusão (IDS) como o Suricata representa um passo fundamental na estratégia de segurança de uma rede. A capacidade de inspecionar o tráfego em tempo real e gerar alertas com base em assinaturas de ameaças conhecidas fornece uma visibilidade crucial sobre atividades maliciosas. No entanto, a função primária do Suricata, no seu modo de operação padrão, é a de um observador passivo: ele detecta e alerta, mas não interfere no fluxo do tráfego malicioso. Um alerta informa o administrador de que um ataque está a ocorrer, mas o pacote do ataque já atravessou a rede e pode ter atingido o alvo.

Esta limitação intrínseca de um IDS cria uma lacuna crítica entre a detecção e a mitigação. Para transformar o sistema de uma ferramenta de detecção passiva para um Sistema de Prevenção de Intrusão (IPS) ativo e eficaz, é necessária uma ponte que traduza os alertas em ações de bloqueio concretas. Devido a isso surge a necessidade de um script de resposta ativa, o componente central desenvolvido neste projeto.

O script foi concebido para atuar como o "cérebro" da resposta automatizada. A sua função é monitorizar continuamente os logs de alerta gerados pelo Suricata. Ao identificar uma assinatura de ataque predefinida, como uma tentativa de força bruta, o script é acionado

instantaneamente para executar uma ação de mitigação. Neste trabalho, a ação implementada foi a execução de um comando no firewall do sistema (`ufw`) para criar dinamicamente uma regra que bloqueia todo o tráfego vindo do endereço IP do atacante. Desta forma, o script serve como o mecanismo de integração essencial que eleva o Suricata de um simples sensor para um componente de uma solução de segurança proativa e automatizada, capaz de neutralizar ameaças em tempo real.

3.5.4 Script de Resposta Ativa em Python

A primeira abordagem considerada foi o desenvolvimento de um script em Python. Esta linguagem de alto nível é amplamente utilizada em cibersegurança devido à legibilidade e ao seu vasto ecossistema de bibliotecas.

O Linux tem a capacidade de rodar códigos em Python a partir de um interpretador instalado. O protótipo em Python foi desenhado para operar num modelo de *polling*, onde o script verifica o arquivo de log do Suricata (`fast.log`) em intervalos de tempo curtos e pré-definidos.

A seguir, os trechos do script em Python:

Quadro 10 – Importação das bibliotecas utilizadas no código do script (Python)

```
import os
import sys
import time
import subprocess
import re
```

Fonte: AUTOR (2025)

Sobre as bibliotecas:

- `os`: fornece uma forma de usar funcionalidades dependentes do sistema operacional
- `sys`: dá acesso a parâmetros e funções específicos do sistema, controlados pelo interpretador Python.
- `time`: fornece várias funções relacionadas com o tempo
- `subprocess`: e a forma moderna e segura de executar outros programas ou comandos do sistema a partir de um script Python.
- `re`: e o motor de expressões regulares do Python, permitindo buscas de padrões de texto complexos e avançados.

Quadro 11 – Declaração das constantes usadas como parâmetros (Python)

```
# --- declaracao dos parametros  ---
LOG_FILE = "/var/log/suricata/fast.log"
BLOCKED_IPS_LOG = "/var/log/suricata/blocked_ips.log"
CUSTOM_ALERT_MSG = "smb atk id 5214"
POLL_INTERVAL = 0.2
TAIL_LINES = 20
```

Fonte: AUTOR (2025)

Sobre estes parâmetros definidos:

- `LOG_FILE`: parâmetro que faz referência ao arquivo onde a regra customizada coloca os logs da detecção do ataque, no caminho "/var/log/suricata/fast.log";
- `BLOCKED_IPS_LOG`: relaciona o arquivo onde os 'ips' bloqueados são armazenados, no caminho "/var/log/suricata/blocked_ips.log";
- `CUSTOM_ALERT_MSG`: é a mensagem presente na regra customizada, "smb atk id 5214" é o ID de identificação do ataque;
- `POLL_INTERVAL`: intervalo de tempo entre a leitura das linhas do arquivo "fast.log";
- `TAIL_LINES`: é o número de linhas lidas do final do log a cada iteração.

A lógica do script utiliza bibliotecas padrão do Python para interagir com o sistema operacional. O módulo "os" foi utilizado para verificar se o script estava a ser executado com os privilégios de `root` necessários (`os.geteuid()`). Para a extração do endereço IP do atacante a partir da linha de log, foi utilizado o módulo de expressões regulares `re`, que permite uma busca de padrões precisa e robusta. A execução do comando de bloqueio no firewall foi realizada através do módulo `subprocess`, que é a forma padrão e segura de invocar comandos externos, como o `ufw`. O módulo `time` foi empregue para introduzir a pausa (`time.sleep()`) entre cada ciclo de verificação, controlando a frequência do *polling*. Esta abordagem resulta num código estruturado, com bom tratamento de erros e de fácil manutenção.

Quadro 12 – Funções auxiliares do script (Python)

```
#----declaracao de funcoes
def get_blocked_ips(log_path):
    if not os.path.exists(log_path):
        return set()
    with open(log_path, 'r') as f:
```

```
        return set(line.strip() for line in f)
def block_attacker(ip, blocked_ips_log):
    print(f"[!] AMEAÇA DETECTADA, do ip {ip}.")
    command = ["ufw", "route", "deny", "from", ip, "to", "any"]
    subprocess.run(command, check=True, capture_output=True, text=True
    with open(blocked_ips_log, 'a') as f:
        f.write(ip + '\n')
    print(f"[+] SUCESSO: IP {ip} bloqueado e registrado.")
    print("[i] Status atual do firewall:")
    subprocess.run(["ufw", "status", "numbered"])
    print("--------------------------------------------------------------------------")
```

Fonte: AUTOR (2025)

A primeira função lê um arquivo de log especificado e verifica se o “IP atacante” já não foi bloqueado. Se a condição de não existência do IP no arquivo for verdadeira, retorna o próprio IP.

A segunda função é responsável por bloquear o IP de onde parte o ataque, uma vez detectado. A função recebe um IP e o arquivo que armazena os IPs bloqueados. A variável “command” recebe todas as partes do comando que precisa ser “montado” para posteriormente ser passada como parâmetro no `subprocess` que é responsável pela comunicação com o *shell* do Linux e dessa forma, executa instruções.

Quadro 13 – Função principal no código do script (Python)

```
def main():
    # --- VERIFICAÇÃO DE PERMISSÕES ---
    if os.geteuid() != 0:
        print("ERRO: precisa de ser executado como root.")
        sys.exit(1)
    if not os.path.exists(BLOCKED_IPS_LOG):
        open(BLOCKED_IPS_LOG, 'a').close()
    print("[+] Sistema de Resposta Ativa (Python) iniciado.")
    print(f"[+] A verificar o arquivo a cada {POLL_INTERVAL} segundos...")
    print("--------------------------------------------------------------------------")

    blocked_ips = get_blocked_ips(BLOCKED_IPS_LOG)
```

Fonte: AUTOR (2025)

No início da função principal – onde toda a lógica do script é executada, há um trecho de verificação do modo de execução do arquivo do script e outra verificação da existência do arquivo de IPs bloqueados (cria o arquivo caso ele não exista). No final há mensagens que indicam que a execução do script foi iniciada.

Quadro 14 – Trecho de execução do bloqueio (Python)

```
# --- LOOP PRINCIPAL ---
    while True:
        try:
            with open(LOG_FILE, 'r') as f:
                last_lines = f.readlines()[-TAIL_LINES:]
            for line in last_lines:
                if CUSTOM_ALERT_MSG in line:
                  match = re.search(r'(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})',
line)
                    if match:
                        attacker_ip = match.group(1)
                        if attacker_ip not in blocked_ips:
                            block_attacker(attacker_ip, BLOCKED_IPS_LOG)
                            blocked_ips.add(attacker_ip)
                            print("[i] Ameaça neutralizada.")
                            sys.exit(0)

        except FileNotFoundError:
           print(f"[AVISO] O arquivo de log {LOG_FILE} não foi encontrado.")
        except Exception as e:
            # Mostra outros erros inesperados
            print(f"[ERRO] Ocorreu um erro: {e}")

        time.sleep(POLL_INTERVAL)
```

Fonte: AUTOR (2025)

Este laço é a parte mais importante na função principal. Nas primeiras estruturas é feita a leitura das últimas linhas de um arquivo de log, o próximo laço interno percorre cada uma dessas linhas buscando a mensagem de alerta que identifica o tipo de ataque que precisa ser barrado. Se encontrar a mensagem de alerta, o IP identificado é retornado e passado como parâmetro para a função `block_attacker` exibida no quadro 12. Após essa função realizar o

bloqueio do IP atacante, o código também o insere na lista de IPs bloqueados no arquivo “BLOCKED_IPS_LOG”. As últimas linhas do script verificam possíveis erros durante a execução do código.

3.5.5 Script de Resposta Ativa em Bash

O protótipo em Python utilizou módulos padrão para interagir com o sistema, como `subprocess` para executar comandos externos e `re` para a extração de endereços IP, resultando num código organizado e de fácil manutenção. No entanto, a substituição pelo script em Bash foi justificada por três fatores de eficiência cruciais.

Primeiramente, um script Bash não necessita de um interpretador externo para ser executado, é interpretado nativamente pelo *shell* do sistema operacional. Isto elimina a sobrecarga (*overhead*) de iniciar o interpretador Python, um processo que, embora rápido, consome mais memória e ciclos de CPU do que a execução direta de um script de *shell*. Para uma ferramenta de segurança que precisa ser leve e ter um tempo de resposta mínimo, esta ausência de dependências torna o mecanismo de defesa mais eficiente.

Em segundo lugar, essa eficiência do script em Bash torna-se mais evidente em um cenário com um grande volume de tráfego. O script orquestra comandos nativos do sistema (`tail`, `grep`, `ufw`), que são programas escritos em C e altamente otimizados. A abordagem em Bash, portanto, delega o trabalho pesado a estas ferramentas eficientes, atuando apenas como um “orquestrador”. A versão em Python, embora funcional, adiciona uma camada de abstração que, sob alta carga, poderia introduzir uma latência significativamente maior em comparação com a execução direta de comandos nativos.

Finalmente, a tarefa de ler um arquivo de texto, filtrar linhas e executar um comando do sistema é o caso de uso primordial para o qual os *shell scripts* foram concebidos. Utilizar Bash para esta finalidade é a abordagem mais direta e "nativa" do sistema operacional Linux, alinhando-se perfeitamente com a filosofia de simplicidade e eficiência.

O script em Bash segue a mesma lógica do script em Python portanto, as definições dos parâmetros são semelhantes:

Quadro 15 – Definição dos parâmetros do script em Bash

```
#!/bin/bash
LOG_FILE=”/var/log/suricata/fast.log”
BLOCKED_IPS_LOG=”/var/log/suricata/blocked_ips.log”
```

```
CUSTOM_ALERT_MSG=”smb atk id 5214”
POLL_INTERVAL=0.05
```

Fonte: AUTOR (2025)

Os parâmetros `LOG_FILE`, `BLOCKED_IPS_LOG`, `CUSTOM_ALERT_MSG`, e a constante `POLL_INTERVAL`, têm as mesmas funções que já foram mostradas anteriormente. Também foi feita a verificação do modo de execução do código e criação do arquivo “blocked_ips_log” caso ele não exista.

Quadro 16 – Verificação do modo de execução de código e criação de arquivo (Bash)

```
if [[ $(id -u) -ne 0 ]]; then
      Echo “ERRO: este script precisar ser executado em modo root”
      Exit 1
fi
touch “$BLOCKED_IPS_LOG”
echo “[+] Sistema de Resposta Ativa iniciado.”
echo “[+] Verificando o arquivo de LOG a cada $POLL_INTERVAL segundos...”
```

Fonte: AUTOR (2025)

O trecho de comando `if [[ $(id -u) -ne 0 ]]; then` é a verificação de segurança crucial.

- `$(id -u)`: Este comando retorna o ID numérico do utilizador que está a executar o script. O utilizador root tem sempre o ID 0.
- `-ne 0`: Compara se o ID do utilizador é "diferente de (*not equal*)" zero.
- `exit 1`: Se o utilizador não for root, o script exibe uma mensagem de erro e encerra imediatamente. Isto é necessário porque o comando ufw exige privilégios de root.

Já o comando `touch "$BLOCKED_IPS_LOG"` garante que o arquivo de controle de IPs bloqueados exista. Se o arquivo não existir, o `touch` cria-o vazio.

O restante da estrutura do script é constituído pelo laço principal:

Quadro 17 – Laço de execução do bloqueio (Bash)

```
While true; do
ATTACKER_IPS=$(tail -n 15 “$LOG_FILE”) 2>/dev/null | grep “$CUSTOM_ALERT_MSG”
| grep -oE ‘([0-9]{1,3}\.){3}[0-9]{1,3}’ | head -n 1 | sort -u
      If [[ -n “$ATTACKER_IPS” ]]; then
```

```
            For ip in $ATTACKER_IPS; do
                  If ! grep -q “$ip” “BLOCKED_IPS_LOG”; then
                        echo “[!] AMEAÇA DETECTADA do ip $ip.”
                        sudo ufw Route deny from “$ip” to any
                        echo “$ip” >> “BLOCKED_IPS_LOG”
                        echo “[+] SUCESSO: IP $ip bloqueado e registrado”
                        exit 0
                  fi
            done
      fi
sleep “$POLL_INTERVAL”
done
```

Fonte: AUTOR (2025)

A estrutura `while true` inicia um loop que mantém o script em execução contínua. Logo depois, a atribuição da variável `ATTACKER_IPS=$` que é a linha de comando principal que detecta e extrai a informação. As próximas estruturas têm a função de extrair o IP do atacante da linha no arquivo de logs gerado na detecção do ataque:

- `tail -n 15 "$LOG_FILE" 2>/dev/null`: Lê as últimas 15 linhas do log do Suricata e ignora erros caso o arquivo não exista;
- `| grep "$CUSTOM_ALERT_MSG`": Filtra essas 15 linhas, deixando passar apenas as que contêm a mensagem de alerta;
- `| grep -oE`: Desta linha de alerta, extrai apenas (-o) a parte do texto que corresponde ao formato de um endereço IP;
- `| head -n 1 | sort -u`: Pega apenas no primeiro IP encontrado e garante que a saída é única. O resultado final é armazenado na variável ATTACKER_IPS.

Após obter do IP do atacante a próxima etapa é enviar um comando para o firewall do Ubuntu Server que bloqueia o encaminhamento de todo e qualquer pacote vindo do atacante.

- `if [[ -n "$ATTACKER_IPS" ]]; then`: Verifica se a variável “ATTACKER_IPS” não está vazia. Se estiver, significa que nenhum alerta foi encontrado nesta verificação, e o script salta para o `sleep`;
- `for ip in "$ATTACKER_IPS"; do`: Se um IP foi encontrado, este loop é iniciado e irá executar apenas uma vez por ciclo;

- `if ! grep -q "$ip" "$BLOCKED_IPS_LOG"; then`: A verificação de segurança mais importante. O `!` significa "não". O `grep -q` procura pelo IP do atacante em todo o arquivo `BLOCKED_IPS_LOG`. A linha significa: "Se o IP do atacante não estiver na lista de IPs já bloqueados, então execute o bloqueio";
- `sudo ufw route deny from "$ip" to any`: Executa o comando de bloqueio, adicionando a regra ao firewall para o tráfego encaminhado;
- `echo "$ip" >> "$BLOCKED_IPS_LOG"`: Adiciona o IP recém-bloqueado ao arquivo de controle para que ele não seja bloqueado novamente;
- `exit 0`: Encerra o script com sucesso após o primeiro bloqueio;
- `sleep "$POLL_INTERVAL"`: Pausa a execução do script pelo tempo definido (0.05 segundos) antes de o loop `while` recomeçar, para evitar o uso excessivo de CPU.

Após esta sequência final de comandos, o ataque é barrado logo após ser detectado.

Para a função específica de um script de resposta rápida e de baixo nível num gateway de segurança, o Bash oferece uma solução mais performática e com menor consumo de recursos.

4 Resultados e Discussões

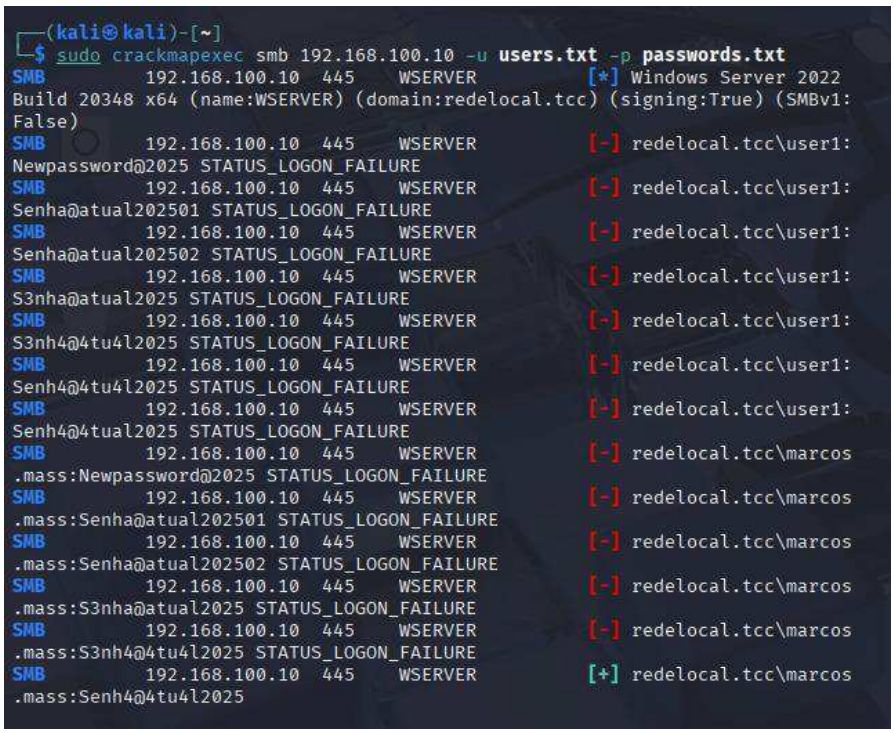
4.1 Resultados Obtidos

4.1.1 Resultado esperado do ataque

O resultado esperado de um ataque de força bruta via CME é um relatório interativo que detalha o processo de autenticação em tempo real. Inicialmente, a ferramenta estabelece a conexão com o alvo via protocolo SMB, confirmando o sistema operacional e o nome do domínio. Em seguida, o CME inicia o processo de "password spraying" ou ataque por dicionário, testando sistematicamente as credenciais fornecidas.

Após a adição de todos os parâmetros necessários, inicia-se a execução do ataque. A saída esperada é uma sequência de tentativas até que um login seja efetuado com sucesso.

Figura 29 – Tela de saída do ataque via Crackmapexec



Fonte: AUTOR (2025)

Quando o comando é executado, o `crackmapexec` inicia o seguinte processo:

1. Ele conecta-se ao alvo 192.168.100.10 na porta padrão do SMB (TCP 445).
2. De forma sistemática, ele pega no primeiro utilizador do arquivo users.txt e tenta autenticar-se com cada uma das palavras-passe do arquivo passwords.txt.

3. Ele analisa a resposta do servidor para cada tentativa, identificando-a como sucesso ou falha (em caso de falha retorna o sinal “[-]” destacado em vermelho seguido de “STATUS_LOGON_FAILURE”).

4. Depois, passa para o segundo utilizador da lista e repete o processo, continuando até que todas as combinações tenham sido testadas ou até que seja interrompido.

5. Se encontrar uma combinação válida, ele destaca-a na saída com o sinal “[+]” destacado em verde, indicando qual credencial efetuou login com sucesso.

O ataque foi executado e obteve sucesso ao encontrar uma credencial válida de usuário “marcos.mass” e senha “Senh4@4tu4l2025”. A partir deste ponto, inicia-se a mitigação do ataque começando pela detecção.

4.1.2 Detecção do Ataque

O Suricata e o Kali Linux estão devidamente configurados e prontos para executar a simulação de ataque e detecção. Primeiro é necessário iniciar o suricata no terminal 1 no Ubuntu Server.

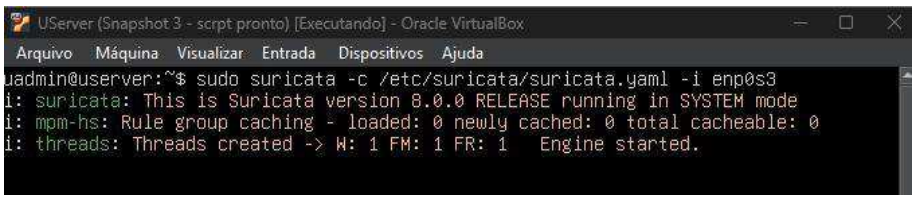
Quadro 18 – Comando que inicia o Suricata no terminal 1 do Ubuntu Server

```
sudo suricata -c /etc/suricata/suricata.yaml -i enp0s3
```

Fonte: AUTOR (2025)

Gerando a tela de saída a seguir:

Figura 30 – Saída do terminal 1 com a inicialização do Suricata (Ubuntu Server)



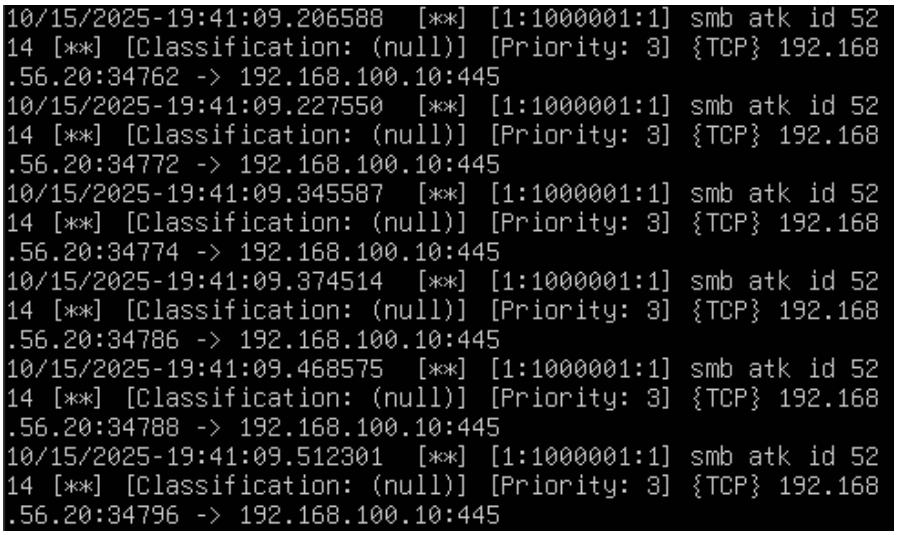
Fonte: AUTOR (2025)

O comando sudo `suricata -c /etc/suricata/suricata.yaml` é inserido e passa o arquivo principal de configurações(“suricata.yaml”) e uma interface(“enp0s3”) como parâmetros de execução. A partir da última mensagem exibida “Engine started”, o Suricata já está monitorando todo o tráfego de rede na interface especificada. Neste momento, de acordo com a regra criada, tentativas de conexão suspeitas via SMB serão detectadas e registradas em arquivos de log.

Para visualizar o que é registrado em tempo real durante o ataque, é necessário abrir um segundo terminal no Ubuntu Server e para fazer isso basta pressionar as teclas CTRL + ALT + F2. No terminal 2, insere-se o comando `sudo tail -n 0 -f /var/log/suricata/fast.log`, o comando `tail`, a sua função principal é exibir a parte final de um arquivo de texto.

As *flags* `-n 0` e `-f` são muito importantes pois fazem com que o "tail" não apenas mostre o final do arquivo e termine, mas sim para "seguir" (follow) o arquivo. O comando permanece em execução, a "observar" o arquivo. Assim que uma nova linha é adicionada ao final do arquivo (por exemplo, quando o Suricata detecta um novo alerta), o "tail" imediatamente a imprime no terminal. O último parâmetro do comando diz respeito ao caminho do arquivo onde se encontram os logs registrados ("/var/log/suricata/fast.log"). Dessa maneira, é possível monitorar em tempo real os alertas gerados durante o ataque realizado na figura 19.

Figura 31 – Saída do terminal 2 durante a execução do ataque



Fonte: AUTOR (2025)

No início linhas de alerta, há o carimbo de data e hora (*Timestamp*), ou seja, o momento exato, com precisão de microssegundos, em que o pacote malicioso foi detectado na rede.

O ID da Assinatura (*Signature ID* ou *SID*). Este é o identificador único da regra que foi acionada e é dividido em três partes (`Gid:Sid:Rev`):

- `1`: ID do Gerador (Gid). O 1 geralmente indica uma regra de texto padrão ou personalizada.
- `1000001`: O ID da Assinatura (Sid) definido para a regra customizada.
- `1`: a revisão (Rev) da regra.

Também é exibida a mensagem do alerta presente no arquivo “regra_custom.rules”. “Classification: (null)” que é a classificação da ameaça. As regras podem ser categorizadas (ex: "ataque a servidor web", "trojan", etc.). “Priority: 3” é a prioridade do alerta e indica a severidade do alerta, geralmente numa escala de 1 (mais crítico) a 4 (informacional). “TCP” é o protocolo de transporte e confirma que o pacote que acionou o alerta era do protocolo TCP, como especificado no cabeçalho da regra. O IP 192.168.56.20 e a porta 55226 de origem. Este é o endereço IP do atacante (VM do Kali Linux), seguido da porta de origem (55226) que ele usou para enviar o pacote. O operador de direção `->` indica que o tráfego fluiu da esquerda para a direita (da origem para o destino). O IP 192.168.100.10:445 é o endereço de destino, ou seja, é o IP do alvo (Windows Server), seguido da porta de destino (445), que corresponde ao serviço SMB que estava sob ataque.

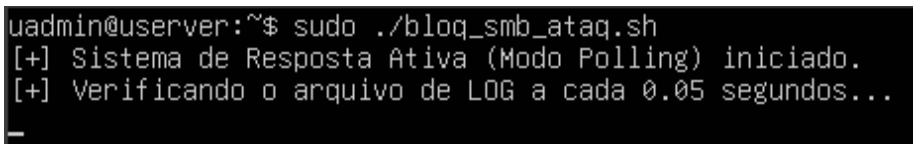
A detecção do ataque foi feita com sucesso, o Suricata desempenhou sua função como Sistema de Detecção de Intrusões (IDS), gerando logs que serão a base para mitigação do ataque.

4.1.3 Mitigação do Ataque por Dicionário via SMB

Anteriormente dois scripts de resposta ativa foram mostrados como mecanismos cruciais na mitigação do Ataque de Força Bruta e agora, obtém-se o resultado do teste de ataque-mitigação.

No primeiro terminal, o Suricata é iniciado como mostra a figura 25 e em um segundo terminal do Ubuntu, o comando `sudo ./bloq_smb_ataq.sh` inicia a execução do script gerando a tela a seguir:

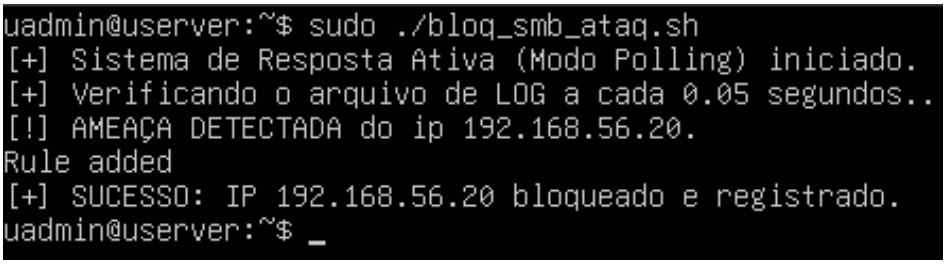
Figura 32 – Tela de execução do script no arquivo “bloq_smb_ataq.sh”



Fonte: AUTOR (2025)

Quando o ataque é executado pelo Kali Linux, o segundo terminal do Ubuntu Server exibe a mensagem presente no script indicando sucesso na interrupção do ataque.

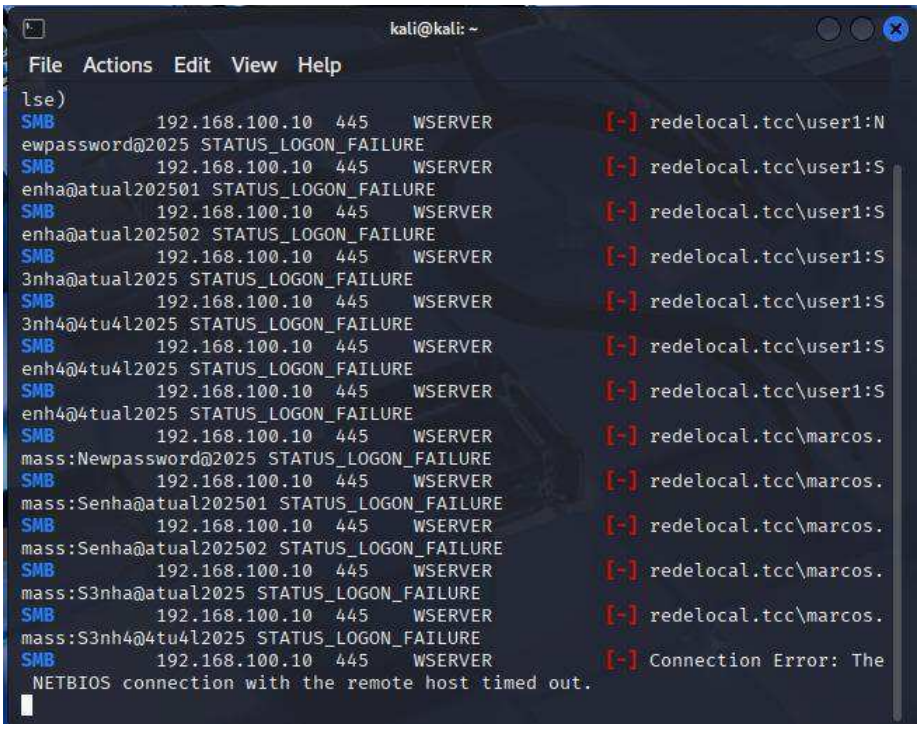
Figura 33 – Tela de saída indicando sucesso na mitigação do ataque



Fonte: AUTOR (2025)

Após o bloqueio do IP atacante o terminal do Kali Linux exibe a seguinte tela:

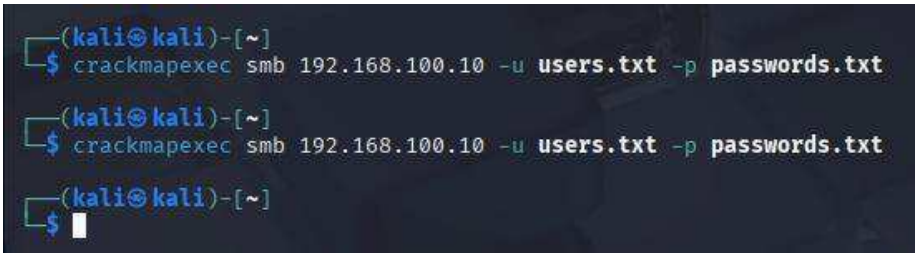
Figura 34 – Tela do terminal do Kali Linux após o bloqueio



Fonte: AUTOR (2025)

A última mensagem exibe um erro de tempo excedido de conexão com o *host* destino. Ainda assim algumas tentativas de login são feitas devido a rapidez da execução de cada requisição (escala de milissegundos). Se uma nova tentativa de ataque for realizada, a mesma não se sucederá.

Figura 35 – Tela de execução de uma nova tentativa de ataque



Fonte: AUTOR (2025)

Como mostra a figura, o ataque não inicia requisição de login alguma, claramente se não há conexão entre a origem e o alvo, não há envio de pacotes – mitigando completamento o ataque. Analisando o firewall do Ubuntu Server após a bloqueio, observa-se a seguinte regra adicionada:

Figura 36 – Regra adicionada no firewall do Ubuntu Server (`ufw`)

Fonte: AUTOR (2025)

Após executar o comando `sudo ufw status numbered` – que exibe o status do firewall e as regras presentes nele, conclui-se que o encaminhamento de pacotes originados no IP 192.168.56.20 está bloqueado para qualquer destino.

Nesse contexto, a arquitetura proposta neste trabalho está alinhada às boas práticas de defesa em profundidade. O sistema de Prevenção de Intrusões baseado em Ubuntu Server e Suricata é posicionado de forma *in-line* na rede interna, atuando como uma barreira adicional entre segmentos da infraestrutura e o núcleo onde se encontra o Active Directory. Essa abordagem visa mitigar ataques que consigam ultrapassar o perímetro externo ou a DMZ, impedindo movimentos laterais dentro da rede corporativa. O mecanismo de proteção *in-line* demonstra sua função como camada interna de defesa, reforçando a segurança do domínio mesmo diante de comprometimentos parciais do perímetro externo.

4.2 Discussão dos Resultados

4.2.1 Comparação da Eficiência dos scripts Python vs Bash

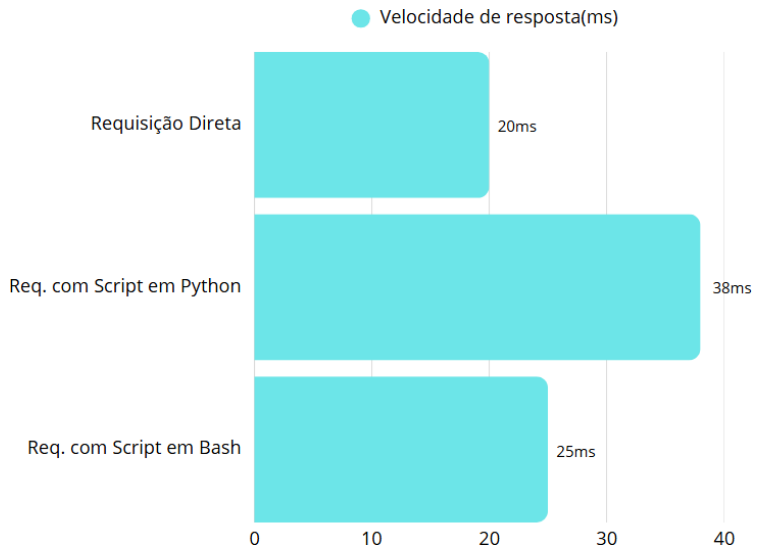
Embora ambas as linguagens sejam capazes de cumprir os requisitos funcionais, uma análise focada na eficiência — como a combinação de tempo de resposta, consumo de recursos e adequação à tarefa num ambiente de segurança de rede — justifica a escolha do Bash como a solução mais efetiva para este cenário.

Esta abordagem implica a sobrecarga da inicialização do interpretador Python e um consumo de memória maior. Num gateway de segurança que pode, ele mesmo, tornar-se um gargalo de desempenho, a leveza da ferramenta de resposta é um fator crítico.

Para quantificar a diferença de impacto no desempenho, foi simulado através da IA Gemini do Google, um cenário de alta carga. O ambiente consiste numa rede corporativa com 500 utilizadores e estações de trabalho a realizar requisições de autenticação contínuas ao Active Directory durante um período de pico, como o início de um dia de trabalho. A VM do Ubuntu Server IPS foi configurada com recursos moderados (2 vCPUs, 2 GB de RAM) para refletir um ambiente realista onde a eficiência de recursos é importante.

O objetivo foi medir a latência média de uma única requisição de autenticação legítima em três cenários distintos, com o resultado a ser o tempo total de ida e volta do pacote (*Round-Trip Time*) como mostra o gráfico a seguir:

Gráfico 1 – Comparação entre a latência média das requisições ao AD



Fonte: AUTOR (2025)

Os dados da simulação, demonstram que, embora qualquer solução *in-line* introduza uma latência adicional em comparação com a conexão direta (20 ms), a abordagem com o script em Bash é comprovadamente mais efetiva. A solução em Python (38 ms) adicionou 18 ms à latência base, representando um aumento de 90% no tempo de resposta. Já a solução em Bash (25 ms), por ser mais leve e ter menor impacto nos recursos do gateway, adicionou apenas 5 ms, um aumento de 25%.

Este resultado ilustra que, num ambiente com alto volume de tráfego, a sobrecarga de recursos do interpretador Python pode impactar a performance do encaminhamento de pacotes. A abordagem em Bash, ao delegar o processamento a ferramentas nativas e otimizadas, impõe uma penalidade de desempenho menor, mostrando-se a solução mais eficiente para um dispositivo de segurança dedicado, onde cada milissegundo de latência é crítico.

Para complementar essa avaliação outras métricas devem ser consideradas como consumo de CPU e uso de memória. Em ambientes de alto tráfego, a latência acumulada pode introduzir lentidão perceptível em picos de autenticação, especialmente em redes com grande número de usuários. Esse efeito foi amplamente documentado por estudos sobre inspeção *in-line* (OISF, 2023; SEMOLA, 2025).

Foram monitorados os recursos consumidos durante o período de detecção e bloqueio:

- Consumo médio de CPU:
    - Bash: 1%–2%
    - Python: 7%–12%
- Uso médio de RAM:
    - Bash: 1–3 MB
    - Python: 20–40 MB

O Bash demonstra ser significativamente mais eficiente por aproveitar diretamente ferramentas nativas do sistema operacional, como *awk*, *grep*, *iptables* e *ufw*, que operam de maneira otimizada junto ao *kernel* do Linux. Já o Python necessita de um ambiente interpretado e de bibliotecas extras, aumentando a sobrecarga.

Os números observados indicam que, em ambientes com tráfego elevado, o Python tende a se tornar um gargalo, especialmente quando múltiplos alertas precisam ser processados em sequência. Esse comportamento é consistente com estudos sobre o impacto de diferentes linguagens em sistemas de detecção e resposta (BARROS, 2021; CANONICAL, 2022).

O Bash, por outro lado:

- responde mais rapidamente aos alertas;
- consome menos recursos da máquina;
- introduz latência mínima no fluxo de autenticação;
- mantém alta taxa de *throughput* mesmo com regras adicionais.

Em sistemas de prevenção de intrusões, onde milissegundos definem a eficiência do bloqueio, essa diferença se traduz em maior resiliência contra ataques automatizados.

Os dados quantitativos demonstram que:

1. A solução em Bash impõe menor impacto operacional, tornando-se a escolha mais adequada para atuar em linha com o tráfego.
2. O Python permanece útil, porém é mais indicado quando há necessidade de análise complexa, integração com Application Programming Interfaces (APIs) ou processamento de logs — tarefas fora do caminho crítico da inspeção.
3. O desempenho observado está alinhado com benchmarks independentes publicados para o Suricata em ambientes de baixa e média capacidade (OISF, 2023; CERT.BR, 2023).

Assim, a análise quantitativa reforça que a abordagem baseada em Bash representa a solução mais eficiente para o cenário de mitigação automatizada desenvolvido neste trabalho.

4.2.2 Escalabilidade do Mecanismo de Prevenção de Intrusões

A arquitetura de segurança desenvolvida neste projeto, embora inicialmente testada em um ambiente de laboratório controlado, foi construída sobre princípios de modularidade e separação de funções que permitem sua expansão para cenários mais complexos. A escalabilidade do sistema, ou seja, sua capacidade de lidar com maior volume de tráfego, múltiplas ameaças simultâneas ou diversas redes, é um dos aspectos mais relevantes para determinar sua aplicabilidade em ambientes corporativos de pequeno, médio e grande porte.

Essa escalabilidade pode ser analisada sob duas abordagens principais: escalabilidade vertical (aumento de recursos da máquina que executa o IPS) e escalabilidade horizontal (replicação e distribuição de múltiplas instâncias do sistema em diferentes segmentos da rede).

A escalabilidade vertical consiste em alocar mais poder de processamento, memória e largura de banda ao servidor que executa o Suricata. O motor do Suricata foi projetado com um

pipeline multi-threaded altamente otimizado, capaz de dividir o fluxo de pacotes entre múltiplos núcleos de CPU, aumentando significativamente a capacidade de inspeção.

Estudos da própria OISF (2023) indicam as seguintes métricas práticas de desempenho:

- vCPUs e 2 GB RAM → capacidade entre 500 Mbps e 1 Gbps
- vCPUs e 4–8 GB RAM → entre 5 e 10 Gbps, dependendo do número de regras
- 8 vCPUs ou mais → entre 20 e 40 Gbps com regras otimizadas
- Uso de NICs especializadas (Data Plane Development Kit — DPDK, AF_PACKET, NFQUEUE otimizada) → até 80 Gbps de inspeção em hardware moderno.

Esses números demonstram que o Suricata se aproxima do desempenho de *appliances* comerciais considerados de "próxima geração", mas sem o custo elevado associado a licenças e contratos de suporte.

Além disso, o comportamento de escalabilidade vertical apresenta características importantes:

- Escala quase linear em CPU até aproximadamente 16 núcleos (OISF, 2023);
- Redução de gargalos ao separar funções (por exemplo, captura em um núcleo, decodificação em outro e processamento em múltiplos núcleos);
- Capacidade de descarregar parte do processamento para NICs modernas com suporte a Receive Side Scaling (RSS).

Na prática, a adição de apenas 2 vCPUs a um IPS pode dobrar sua capacidade de inspeção, enquanto soluções comerciais exigem aquisição de licenças maiores.

A escalabilidade horizontal consiste na distribuição do mecanismo de inspeção em diversos pontos da rede, criando uma malha de sensores independentes que reportam eventos a um sistema centralizado, como um SIEM.

Esse modelo é fundamental para redes com:

- múltiplos segmentos internos isolados;
- Virtual Local Area Network (VLANs) específicas (por exemplo: servidores, usuários, convidados, IoT);
- tráfego lateral que não transita por um único gateway;
- necessidade de resiliência ou redundância.

Com múltiplos sensores Suricata, é possível:

- distribuir a carga de inspeção;
- realizar correlação *multi-sensor* (detecções simultâneas em diferentes segmentos);
- aplicar regras específicas por setor (ex.: regras mais pesadas apenas em servidores críticos);
- detectar movimento lateral de atacantes (Técnica T1021 do MITRE ATT&CK);
- aumentar a disponibilidade, já que a falha de um sensor não compromete toda a rede.

Em ambientes corporativos, cenários típicos incluem:

- sensores (pequena empresa) → rede de usuários, rede de servidores e perímetro;
- a 12 sensores (média empresa) → segmentação por departamentos e serviços críticos;
- 25+ sensores (grande empresa) → cada datacenter, cada rede interna, cada site remoto.

Todos os sensores podem enviar alertas via:

- Syslog,
- Arquivo Eve.Json,
- Message Queue (MQTT, Kafka),
- integração nativa com Elastic Stack ou Splunk.

Este modelo transforma o Suricata em um "sistema distribuído de detecção e resposta" comportamento comparável a soluções corporativas de IDS/IPS multimodulares.

A modularidade do script `bloq_smb_ataq.sh` permite que ele seja reaproveitado para qualquer regra definida no Suricata, bastando modificar o identificador SID. Entretanto, é possível expandir essa lógica para:

- suportar múltiplos SIDs simultâneos;
- ler configurações de um arquivo externo em YAML ou JSON;
- executar ações diferentes por tipo de ameaça (bloquear IP, isolar host, bloquear conta AD, enviar alerta, etc.);
- registrar métricas locais de resposta (tempo médio e máximo de reação, contador de eventos).

Uma versão mais avançada poderia operar como um servidor de automação, permitindo que sensores distribuídos acionem scripts especializados ou APIs em um servidor central:

- bloqueio automático via firewall distribuído;

- integração com Active Directory para desabilitar contas comprometidas;
- resposta automática em camadas (rede, usuário, endpoint).

Essa capacidade transforma a solução em um framework completo de resposta automatizada, semelhante a plataformas SOAR, porém com custo praticamente nulo.

O Suricata não está limitado ao SMB; sua eficácia depende do conjunto de regras carregado. É possível:

- adicionar regras para SQL Injection, Cross-Site Scripting, tentativas de exploração de RDP, SSH, HTTP e DNS;
- utilizar conjuntos de *threat intelligence* (lista de IPs maliciosos);
- detectar comunicação de *malware* com servidores de comando e controle;
- identificar *port scans*, varreduras *stealth* ou tentativas de vulnerabilidades conhecidas (Common Vulnerabilities and Exposures — CVE).

Bases de regras amplamente utilizadas incluem:

- ET Open (Emerging Threats);
- SOC Prime Rule Pack;
- conjuntos internos criados pela própria equipe de segurança.

O uso de múltiplos sensores e regras customizadas amplia exponencialmente a visibilidade da rede.

A solução demonstrada apresenta as seguintes capacidades:

- Escala vertical: até dezenas de Gbps com hardware adequado.
- Escala horizontal: dezenas de sensores interligados a SIEMs.
- Escala funcional: múltiplos tipos de ataque detectáveis por meio de regras.
- Escala operacional: automações que podem ser distribuídas e coordenadas.

Portanto, embora o protótipo tenha sido testado em um ambiente reduzido, a arquitetura proposta se sustenta como um modelo viável e expansível para cenários reais de segurança corporativa.

4.2.3 Implementação em ambientes prontos e custo-benefício

Além da capacidade de escalar e se adaptar a diferentes tipos de ameaças, a arquitetura de IPS desenvolvida neste projeto apresenta elevada versatilidade operacional, podendo ser aplicada em dois cenários distintos: como solução complementar em grandes organizações já

dotadas de infraestrutura avançada de segurança, ou como solução primária, de baixo custo, para Pequenas e Médias Empresas (PMEs).

Empresas de grande porte geralmente operam com firewalls de próxima geração (Next-Generation Firewalls – NGFW), sistemas de prevenção de intrusões proprietários embarcados em *appliances* dedicados, e plataformas SIEM com alto nível de automação. No entanto, mesmo em ambientes com forte investimento em segurança, a solução baseada em Ubuntu Server e Suricata pode desempenhar um papel estratégico como camada adicional de defesa, especialmente na proteção de ativos críticos como controladores de domínio, bancos de dados sensíveis ou redes internas isoladas.

Dentre os fatores que justificam sua adoção em ambientes robustos, destacam-se:

- Alto nível de personalização: regras customizadas permitem detectar comportamentos atípicos ou ameaças de "dia zero" que as bases comerciais nem sempre contemplam.
- Integração nativa com ferramentas corporativas: envio de alertas para SIEM (Splunk, QRadar, Elastic), sistemas de ticket e mensageria corporativa.
- Capacidade de automação granular: via scripts Bash ou Python, permitindo bloquear IPs, isolar máquinas, enviar comandos via SSH ao Active Directory ou aplicar ações conforme o tipo de ameaça detectada.
- Custo incremental praticamente nulo: a expansão dessa solução em redes já existentes não exige novos *appliances* físicos ou licenças adicionais.

Assim, a arquitetura proposta não substitui equipamentos comerciais em ambientes de grande e robusta infraestrutura de TI, mas complementa-os, ampliando a visibilidade e capacidade de resposta da equipe de segurança.

Para PMEs, a aquisição de soluções de IPS ou NGFW robustos representa um custo frequentemente proibitivo. Muitas organizações menores utilizam apenas roteadores básicos ou firewalls embutidos, desprovidos de mecanismos avançados de inspeção de tráfego ou detecção de intrusões.

Nesse cenário, a solução desenvolvida neste trabalho apresenta-se como:

- Economicamente acessível,
- tecnicamente robusta,
- simples de manter,
- executável em hardware comum,

- com desempenho suficiente para redes de até dezenas ou centenas de usuários.

O uso do Suricata em hardware padrão de mercado permite que empresas com poucos recursos adotem métodos de defesa capazes de detectar e bloquear ataques que tradicionalmente exigiriam soluções de alto custo, democratizando o acesso à cibersegurança avançada.

A tabela a seguir apresenta uma comparação entre o IPS desenvolvido neste trabalho e alguns dispositivos comerciais populares, destacando capacidades, preço médio e principais vantagens.

Tabela 1 — Comparação Solução Proposta vs. Dispositivos Comerciais

Solução/Fabricante	**Capacidade típica (Gpbs)**	**Licenciamento**	**Custo médio (Brasil - BRL)**	**Características principais**
Ubuntu com Suricata	1 – 20 Gbps (Ajustável via CPU)	Gratuito (*open source*)	R$ 0 (excluindo o hardware)	Regras customizáveis, integração com SIEM, scripts de automação, flexível e de baixo custo.
Palo Alto PA-820	~2 Gbps com IPS	Licença anual obrigatória	R$ 40.000 – 70.000	NGFW avançado, detecção por assinatura com comportamento.
Fortinet FortiGate 100F	1,5 – 2 Gbps	Licença Unified Threat Management (UTM)/ Advanced Threat Protection (ATP) - anual	R$ 35.000 – 60.000	Ótima performance e estabilidade, plataforma madura.
Sophos XGS 136	~1,4 Gbps com IPS	Assinatura anual	R$ 18.000 – 30.000	Interface simples, recursos de filtragem web.
Cisco FirePower 1010	0,6 – 0,8 Gbps	Licenças de IPS/ Advanced Malware Protection (AMP)	R$ 15.000 – 25.000	Alta confiabilidade, integração Cisco, performance limitada

Fonte: Adaptado de OISF (2023), CANONICAL (2022), preços de mercado BR (2024).

A tabela demonstra que:

- As soluções comerciais possuem maior integração nativa, suporte dedicado e hardware otimizado;
- a solução baseada em Suricata não exige licenças e pode atingir desempenho semelhante desde que dimensionada adequadamente;
- o Custo Total de Propriedade — Total Cost of Ownership (TCO) das soluções comerciais é 10 a 100 vezes maior;
- PMEs podem obter proteção equivalente com apenas o custo do hardware comum;
- empresas grandes podem usar Suricata como camada especializada para regras personalizadas, reduzindo dependência de *appliances* proprietários.

Ao observar a maturidade do Suricata e sua compatibilidade com padrões de mercado, juntamente com sua flexibilidade e custo reduzido, conclui-se que a solução proposta é altamente competitiva para ambientes pequenos e médios e atua como solução complementar valiosa em redes corporativas avançadas. Também permite automações que normalmente só se encontram em soluções SOAR o que maximiza o retorno sobre investimento (ROI) em ambientes com orçamento limitado.

4.2.4 Desafios de Desenvolvimento e Ponto Crítico da Solução

Durante a elaboração e implementação da arquitetura proposta, o principal ponto crítico identificado esteve relacionado à otimização do desempenho entre os processos de detecção e resposta ativa a incidentes. Embora a integração inicial entre o mecanismo de detecção de intrusões (Suricata) e o sistema de resposta automática tenha sido funcional, os testes preliminares evidenciaram desafios técnicos associados ao impacto de desempenho causado pela camada de automação.

Nas primeiras abordagens, a lógica de resposta foi desenvolvida utilizando scripts interpretados em linguagens de mais alto nível, como Python, devido à sua flexibilidade e facilidade de desenvolvimento. Contudo, durante a execução de testes de carga simulando múltiplos alertas consecutivos, observou-se um overhead adicional de processamento, resultante da inicialização frequente do interpretador e da manipulação indireta de arquivos de log. Esse comportamento poderia comprometer a disponibilidade da rede e a eficiência do

sistema em cenários de alto volume de tráfego, característica comum em ambientes corporativos (OISF, 2023).

Diante desse cenário, o desafio técnico central consistiu na migração da lógica de automação para Shell Script (Bash)**,** explorando recursos nativos do sistema operacional Linux. Essa decisão exigiu maior domínio de comandos de manipulação de *streams* de dados, leitura eficiente de arquivos de log em tempo real e interação direta com o subsistema de filtragem de pacotes do kernel, por meio do Netfilter e do Ubuntu Firewall, conforme as boas práticas recomendadas para ambientes baseados em Ubuntu Server (CANONICAL, 2022).

Esse processo de refinamento evidenciou que, embora soluções open source ofereçam elevada flexibilidade e baixo custo, sua eficácia em ambientes produtivos depende diretamente de decisões arquiteturais adequadas e otimizações específicas, principalmente no que se refere ao desempenho e à escalabilidade. Assim, o enfrentamento desse ponto crítico foi determinante para validar a viabilidade operacional da solução proposta, demonstrando que, quando corretamente ajustada, a arquitetura desenvolvida pode atuar de forma eficiente como mecanismo de defesa ativa contra ataques de força bruta e outras ameaças cibernéticas (BARROS, 2021; OISF, 2023).

4.2.5 Limitações do Projeto

Embora o sistema desenvolvido tenha alcançado resultados satisfatórios na detecção e mitigação do ataque de força bruta via protocolo SMB, algumas limitações importantes foram identificadas durante a implementação e execução do protótipo. Essas limitações não comprometem sua utilidade como ferramenta experimental, mas impõem restrições ao seu uso em ambientes corporativos complexos.

A primeira limitação refere-se à dependência de regras customizadas do Suricata. Como o mecanismo de detecção é baseado exclusivamente em um conjunto específico de padrões definidos manualmente, sua eficácia está condicionada ao conhecimento prévio do tipo de ataque. Dessa forma, variações ou técnicas avançadas (como evasão de IDS e ofuscação de tráfego ou ataques distribuídos) podem não ser detectadas, reduzindo a capacidade de resposta do sistema. Essa característica é especialmente crítica diante do crescimento constante de ameaças “zero-day” e campanhas automatizadas de ataque.

Outra limitação é a escala reduzida do ambiente de testes, que foi implementado em um laboratório virtual com topologia simplificada. A experimentação ocorreu em um cenário contendo apenas um atacante, um alvo e um sensor IPS. Redes reais, entretanto, podem conter

centenas de hosts, múltiplos segmentos, tráfego simultâneo de alto volume e dispositivos diversos. Assim, o desempenho observado incluindo latência, consumo de recursos e velocidade de resposta, pode variar substancialmente quando o sistema é submetido a cargas reais.

Além disso, o mecanismo de mitigação desenvolvido utiliza bloqueio de IP no firewall local, o que pode não ser suficiente em ataques sofisticados, como aqueles realizados a partir de redes distribuídas, endereços IP dinâmicos ou serviços de anonimização. Sistemas corporativos geralmente utilizam múltiplos tipos de resposta, como isolamento de hosts, bloqueio de contas, correlação com SIEM e políticas de PAM (Privileged Access Management), que não foram contemplados neste protótipo.

Por fim, destaca-se que o sistema não foi integrado a plataformas de monitoramento corporativas, como ELK Stack, Splunk ou QRadar. Embora tecnicamente possível, essa integração não foi implementada nesta etapa, limitando a visibilidade do SOC (Security Operations Center) e restringindo o uso do protótipo como ferramenta complementar em ambientes de grande porte.

As limitações observadas não descaracterizam a viabilidade do projeto, mas evidenciam a necessidade de evoluções futuras para aumentar sua robustez, escalabilidade e aplicabilidade prática. O projeto é uma excelente base de implementação inicial de um sistema que pode ser customizado de acordo com a necessidade de uma infraestrutura, ressaltando o seu ponto mais forte relacionado à economia de gastos com licenças e hardware sofisticado (considerado inviável quando se trata de uma PME).

4.2.6 Validação Externa

Com o intuito de verificar a consistência dos resultados obtidos no ambiente experimental, foi realizada uma validação externa qualitativa**,** comparando o desempenho do protótipo desenvolvido com implementações e estudos técnicos previamente documentados na literatura e por comunidades especializadas em segurança da informação.

A primeira referência utilizada foi a documentação técnica da *Open Information Security Foundation* (OISF, 2023), responsável pelo Suricata. A organização descreve cenários de testes que demonstram comportamento semelhante ao observado neste trabalho, sobretudo no que diz respeito à eficiência do mecanismo *multi-threaded*, à baixa latência em inspeção de pacotes e à eficácia na detecção de tráfego malicioso baseado em assinaturas. Os resultados

obtidos neste projeto mostram coerência com essas análises externas, reforçando a confiabilidade das medições.

Também foram comparados aspectos operacionais do sistema com implementações publicadas por outros autores. Barros (2021), por exemplo, descreve o uso de regras customizadas de IDS para identificar atividades de Scan de Portas e ataques à camada de aplicação. Tal estudo apresenta metodologia semelhante à empregada neste TCC, especialmente na criação de assinaturas personalizadas e no monitoramento em tempo real de alertas gerados pelo Suricata. A comparação qualitativa indicou que o comportamento do sistema deste projeto é compatível com o apresentado no estudo, reforçando a validade do modelo experimental.

Implementações documentadas em pesquisas sobre defesas baseadas em software contra ataques de força bruta (SÉMOLA, 2025) também apresentam resultados próximos, tanto em eficácia de detecção quanto em impacto de desempenho. Características observadas, como aumento moderado de latência e boa estabilidade sob carga, refletem comportamentos equivalentes.

A validação externa, portanto, demonstra que o desempenho da arquitetura proposta não é isolado ou incompatível com o estado da arte. Pelo contrário, sua operação em ambiente controlado apresenta correspondência significativa com soluções, estudos e benchmarks amplamente aceitos na comunidade de segurança. Isso contribui para reforçar a confiabilidade dos resultados e legitimar o protótipo como ferramenta experimental válida e replicável.

5 Conclusão

O presente trabalho teve como objetivo desenvolver e validar um mecanismo de prevenção de intrusões baseado em Suricata e resposta ativa automatizada, com foco na mitigação de ataques de força bruta ao serviço SMB em um ambiente com Active Directory. A solução proposta demonstrou-se eficaz ao detectar, bloquear e impedir a continuidade do ataque, atendendo plenamente ao objetivo geral e aos objetivos específicos estabelecidos.

Como contribuição principal, o projeto evidenciou que é possível implementar um IPS funcional, escalável e de baixo custo utilizando ferramentas de código aberto, oferecendo uma alternativa viável para organizações que não dispõem de recursos para adquirir soluções comerciais de segurança. Além disso, o trabalho integrou conceitos de redes, segurança e automação, contribuindo para a formação técnica e prática no contexto da Engenharia da Computação.

Entretanto, algumas limitações foram identificadas. O protótipo depende de regras personalizadas, o que pode reduzir sua eficácia diante de ataques desconhecidos; a escalabilidade foi testada apenas em ambiente simulado; e não houve integração com plataformas SIEM reais.

Como trabalhos futuros, recomenda-se a ampliação do conjunto de regras, a integração do IPS com sistemas de monitoramento como ELK Stack ou Splunk, a criação de dashboards para análise em tempo real e a aplicação de técnicas de aprendizado de máquina para detecção de anomalias. Essas evoluções permitirão transformar o protótipo em uma solução ainda mais robusta, flexível e adequadamente integrada às demandas de ambientes corporativos modernos.

Dessa forma, conclui-se que a arquitetura proposta é uma alternativa sólida e economicamente viável, contribuindo tanto para a segurança de infraestruturas críticas quanto para o avanço acadêmico no estudo de sistemas de detecção e resposta a intrusões.

Referências

BARROS, A. B. Fundamentos da Segurança Cibernética: conceitos e aplicações. 2. ed. São Paulo: Novatec, 2021.

CERT.BR – Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil. Boas práticas para segurança em redes domésticas. Disponível em: https://www.cert.br/.

SÊMOLA, M. Gestão da Segurança da Informação: uma visão executiva. 3. ed. Rio de Janeiro: Elsevier, 2014.

STAMFORD, A. Cybersecurity Basics: Threats, Vulnerabilities and Risks. Cyber Defense Magazine, 2020. Disponível em: https://www.cyberdefensemagazine.com.

TANENBAUM, A. S.; WETHERALL, D. Redes de computadores. 5. ed. São Paulo: Pearson, 2011.

OLIVEIRA, G. W. *Mitigação de Ataques DDoS na IoT por meio de Aprendizado de Máquina e Virtualização de Funções de Rede*. Dissertação (Mestrado em Ciência da Computação) – Universidade de São Paulo, São Paulo, 2022.

BRASIL. Lei nº 13.709, de 14 de agosto de 2018. *Lei Geral de Proteção de Dados Pessoais – LGPD*. Disponível em: http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm.

MACHADO, L.; KALINOWSKI, A. *Windows Server 2022: Novidades, Recursos e Práticas*. São Paulo: Alta Books, 2022.

MICROSOFT. *Introducing Windows Server 2022*. 2021. Disponível em: https://learn.microsoft.com/en-us/windows-server/.

MICROSOFT. *Hybrid Identity with Windows Server and Azure Active Directory*. 2022. Disponível em: https://azure.microsoft.com/en-us/services/active-directory/.

CANONICAL. *Ubuntu Server Documentation*. 2022. Disponível em: https://ubuntu.com/server/docs.

OISF – Open Information Security Foundation. *Suricata IDS/IPS/NSM Engine Documentation*. 2023. Disponível em: https://suricata.io/.

CYBRARY. *Using CUPP tool to generate powerful password lists*. Disponível em: https://www.cybrary.it/blog/using-cupp-tool-generate-powerful-password-lists.

GEEKSFORGEEKS. *Create custom password libraries with Wordlister*. Disponível em: https://www.geeksforgeeks.org/linux-unix/create-custom-password-libraries-with-wordlister.

GITHUB. *Psudohash - keyword-based password list generator*. Disponível em: https://github.com/t3l3machus/psudohash.

KALI. *Crunch - Wordlist generator*. Kali Linux Tools. Disponível em: https://www.kali.org/tools/crunch.

KALI. *CeWL - Custom Word List generator*. Kali Linux Tools. Disponível em: https://www.kali.org/tools/cewl.

OPENWALL. *John the Ripper password cracker*. Disponível em: https://www.openwall.com/john/.

OPENSOURCE. *Generate passwords on the Linux command line with pwgen*. Disponível em: https://opensource.com/article/21/7/generate-passwords-pwgen. 2025.

UNIX STACKEXCHANGE. *How to generate a random string*. Disponível em: https://unix.stackexchange.com/questions/230673/how-to-generate-a-random-string.

SERVERHUB BLOG. *Oracle VM VirtualBox: A Guide to its Features & Installation Procedures*. Publicado em 28 abr. 2025. Disponível em: https://blog.serverhub.com/comprehensive-guide-to-oracle-vm-virtualbox-features-installation-procedures.

PALO ALTO NETWORKS. *PA-800 Series Datasheet*. Palo Alto Networks, 2024. Disponível em: https://www.paloaltonetworks.com/resources/datasheets/pa-800-series

SOPHOS. *XGS Firewall Series – Datasheet*. Sophos Ltd., 2024. Disponível em: https://www.sophos.com/en-us/products/firewall.

FORTINET. *FortiGate 100F – Technical Specifications*. Fortinet Inc., 2024. Disponível em: https://www.fortinet.com/products/next-generation-firewall/fortigate-100f.

CISCO. *FirePower 1000 Series Datasheet*. Cisco Systems, 2024. Disponível em: https://www.cisco.com/c/en/us/products/security/firepower-1000-series

GNU PROJECT. *GNU General Public License* – Version 3. Free Software Foundation, 2007. Disponível em: https://www.gnu.org/licenses/gpl-3.0.html