UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS FERREIRA LIMA

ANÁLISE COMPARATIVA DAS FERRAMENTAS IDS OPEN SOURCE SNORT, SURICATA E ZEEK PARA DETECÇÃO DE ATAQUES EM REDES IOT

QUIXADÁ

2026

LUCAS FERREIRA LIMA

ANÁLISE COMPARATIVA DAS FERRAMENTAS IDS OPEN SOURCE SNORT, SURICATA E ZEEK PARA DETECÇÃO DE ATAQUES EM REDES IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Michel Sales Bonfim.

QUIXADÁ

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

L698a Lima, Lucas Ferreira.
Análise comparativa das ferramentas IDS Open Source Snort, Suriacta e Zeek para detecção de ataques em redes IoT / Lucas Ferreira Lima. – 2026.
76 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2026.
Orientação: Prof. Dr. Michel Sales Bonfim.

1. Detecção de Intrusão. 2. IDS Open Source. 3. Segurança IoT. 4. EdgeIIoTset. I. Título.
CDD 004

LUCAS FERREIRA LIMA

ANÁLISE COMPARATIVA DAS FERRAMENTAS IDS OPEN SOURCE SNORT, SURICATA E ZEEK PARA DETECÇÃO DE ATAQUES EM REDES IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 22/01/2026.

BANCA EXAMINADORA

Prof. Dr. Michel Sales Bonfim (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Roberto Cabral Rabêlo Filho
Universidade Federal do Ceará (UFC)

Prof. Dr. Jeandro de Mesquita Bezerra
Universidade Federal do Ceará (UFC)

Dedico este trabalho a mim mesmo, por ter persistido e chegado até aqui, mesmo diante das dificuldades e incertezas enfrentadas ao longo da jornada acadêmica. Apesar do apoio de familiares e amigos, em muitos momentos foi necessário continuar acreditando no meu potencial e buscando sempre superar meus próprios limites.

AGRADECIMENTOS

Agradeço primeiramente à minha mãe, Claudiana, por todo o amor e apoio incondicional durante toda a minha vida e, especialmente, nesta jornada acadêmica. Seu incentivo desde o início foi a maior motivação para que eu pudesse seguir em frente. Não teria chegado até aqui sem isso.

Agradeço ao meu irmão, Pedro Luiz, e ao padrasto, Adriano, pelo suporte emocional durante esses anos de estudo e distância. Saber que vocês estavam ao lado da minha mãe me trouxe a tranquilidade necessária para seguir esse caminho.

Ao Prof. Dr. Michel Sales Bonfim, pela excelente orientação, paciência e incentivo ao longo da realização deste trabalho. Em muitos momentos, não fui o orientando mais responsável e organizado, mas sua compreensão e apoio foram fundamentais para a conclusão deste trabalho.

Aos professores integrantes da banca examinadora, Prof. Dr. Roberto Cabral Rabêlo Filho e Prof. Dr. Jeandro de Mesquita Bezerra, pelas valiosas contribuições e sugestões que possibilitaram o aprimoramento deste trabalho. Além de colaborarem diretamente para a consolidação da proposta, o Prof. Jeandro atuou como meu primeiro orientador em um trabalho que não pôde ser concluído à época, mas cuja experiência representou um aprendizado fundamental e serviu de base para a maturidade acadêmica alcançada neste estudo. Já o Prof. Roberto teve papel decisivo ao despertar meu interesse pela área de segurança da informação, por meio da disciplina de Auditoria e Segurança de Sistemas de Informação.

Por fim, agradeço aos meus amigos e colegas, que estiveram tanto ao meu lado na jornada acadêmica quanto na vida pessoal. A mera presença de vocês fez toda a diferença para que eu pudesse chegar até aqui e não desistisse nos momentos difíceis. Cada uma das risadas, conversas vazias e momentos de descontração foi essencial para que eu pudesse manter o equilíbrio necessário para concluir este trabalho.

Eu posso aceitar a falha, todos falham em alguma coisa. Mas não posso aceitar não tentar.
(JORDAN, Michael.)

RESUMO

O crescimento exponencial de dispositivos da Internet das Coisas (IoT) tem ampliado significativamente a superfície de ataque em infraestruturas críticas, tornando os sistemas de detecção de intrusão (IDS) componentes indispensáveis para a segurança de redes modernas. Esta pesquisa apresenta uma análise comparativa das ferramentas *open source* Snort, Suricata e Zeek, avaliando sua eficácia contra uma grupo abrangente de ameaças, incluindo Ataque de Negação de Serviço Distribuído (DDoS), *Port Scanning*, *Ransomware* e injeções Web, provenientes do *dataset Edge-IIoTset*, com foco exclusivo na análise de tráfego malicioso. Para a condução dos experimentos, utilizou-se o *framework* Dalton para orquestração de contêineres, garantindo o isolamento e a reprodutibilidade dos testes. Como diferencial metodológico, foram desenvolvidos e executados *scripts* de *parsing* customizados, essenciais para a normalização dos logs heterogêneos e a extração precisa de métricas de detecção e consumo de recursos. Os resultados evidenciaram perfis distintos: o Snort obteve a maior precisão (91,5%), ideal para minimização de falsos positivos, mas falhou na detecção de ataques complexos, o Zeek alcançou o maior *recall* (94,0%) com consumo de CPU inferior a 2%, destacando-se pela eficiência e o Suricata apresentou o melhor equilíbrio (F1-Score de 77,6%) e robustez em ataques volumétricos. Portanto, conclui-se que o Suricata é a solução mais indicada para *gateways* IoT, devido à sua resiliência e capacidade de detecção profunda, enquanto o Zeek recomenda-se para sensores IoT com severas restrições de *hardware*, onde a visibilidade leve é prioritária sobre o bloqueio ativo.

Palavras-chave: Detecção de Intrusão; IDS Open Source; Segurança IoT; EdgeIIoTset.

ABSTRACT

The exponential growth of Internet of Things (IoT) devices has significantly expanded the attack surface of critical infrastructures, making intrusion detection systems (IDS) indispensable components for the security of modern networks. This research presents a comparative analysis of the open-source tools Snort, Suricata, and Zeek, evaluating their effectiveness against a comprehensive set of threats, including DDoS, *Port Scanning*, *Ransomware*, and Web injection attacks, derived from the *Edge-IIoTset* dataset, with an exclusive focus on malicious traffic analysis. To conduct the experiments, the Dalton framework was employed for container orchestration, ensuring test isolation and reproducibility. As a methodological contribution, custom parsing scripts were developed and executed, which were essential for normalizing heterogeneous logs and accurately extracting detection metrics and resource consumption data. The results revealed distinct operational profiles: Snort achieved the highest precision (91.5%), making it suitable for minimizing false positives but ineffective in detecting complex attacks; Zeek attained the highest recall (94.0%) while maintaining CPU usage below 2%, standing out for its efficiency; and Suricata provided the best overall balance, with an F1-Score of 77.6%, in addition to demonstrating robustness under volumetric attack scenarios. Therefore, it is concluded that Suricata is the most suitable solution for IoT gateways due to its resilience and deep detection capabilities, whereas Zeek is recommended for IoT sensors with severe hardware constraints, where lightweight visibility is prioritized over active blocking.

Keywords: Intrusion Detection; Open Source IDS; IoT Security; Edge-IIoTset.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

ACC	Acurácia
DDoS	Ataque de Negação de Serviço Distribuído
DL	Aprendizado Profundo
DoS	Ataque de Negação de Serviço
ET	Emerging Threats
F1	F1-Score
FTP	File Transfer Protocol
HIDS	Sistema de Detecção de Intrusão Baseado em Host
HIPS	Sistema de Prevenção de Intrusão Baseado em Host
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IDS	Sistema de Detecção de Intrusão
IIoT	Internet Industrial das Coisas
IoT	Internet das Coisas
IoT-A	Internet das Coisas - Arquitetura
IPS	Sistema de Prevenção de Intrusão
LLM	Large Language Model
MQTT	Message Queuing Telemetry Transport
NIDS	Sistema de Detecção de Intrusão Baseado em Rede
NIPS	Sistema de Prevenção de Intrusão Baseado em Rede
PCAP	Packet Capture
PRC	Precisão
REC	Recall
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol

SUMÁRIO

1 INTRODUÇÃO

Internet of Things (IoT), ou Internet das Coisas, é um termo cunhado pelo pesquisador britânico Kevin Ashton em 1999. Ele usou o conceito para descrever a conexão de objetos físicos através de redes existentes, permitindo integração direta sem intervenção humana para maior eficiência (Gokhale *et al.*, 2018). No cenário atual, a IoT destaca-se pela adaptação a diversas demandas, sendo aplicada em saúde, logística, automação residencial, agricultura de precisão, cidades inteligentes, monitoramento industrial e gestão energética (Atzori *et al.*, 2010).

Segundo a IOT ANALYTICS (2023), o número de dispositivos IoT conectados globalmente atingiu 16,6 bilhões em 2023, com projeção de 40 bilhões até 2030. Esse crescimento mostra como a tecnologia transformou o cotidiano, desde aplicações domésticas como controle climático e segurança até monitoramento de saúde por *wearables*. Na mobilidade urbana, possibilita veículos autônomos e rodovias inteligentes, enquanto na indústria revoluciona processos com robôs conectados. A IoT expande-se para serviços hoteleiros com robôs assistentes e avança para implantes humanos inteligentes, criando ecossistemas hiperconectados (Moraes; Hayashi, 2021).

Se por um lado a IoT traz eficiência e conveniência, por outro eleva exponencialmente os riscos relacionados à segurança cibernética e proteção de dados pessoais, conforme demonstrado por Chicarino *et al.* (2017). Devido à sua heterogeneidade, os dispositivos IoT frequentemente utilizam sistemas operacionais baseados em Linux, que permitem o aproveitamento completo de seus recursos. No entanto, essa característica também torna esses dispositivos vulneráveis a explorações de falhas típicas desse ambiente, como controles inadequados no servidor, ausência de criptografia, proteção binária insuficiente, implementações deficientes de autenticação e autorização, além do uso indevido de serviços de rede (Seralathan *et al.*, 2018). Além dessas vulnerabilidades, existem *malwares* específicos para IoT capazes de causar danos significativos, incluindo o roubo de informações sensíveis e a formação de extensas redes de dispositivos comprometidos (botnets). Um exemplo é o Mirai, *malware* especializado em Ataque de Negação de Serviço Distribuído (DDoS) que explora credenciais padrão para infectar dispositivos inteligentes e incorporá-los a sua botnet (Güven *et al.*, 2023).

Considerando os riscos de segurança em redes IoT, os Sistema de Detecção de Intrusão (IDS) emergiram como uma solução proativa, monitorando continuamente o tráfego da rede para identificar e mitigar atividades suspeitas (Amrullah, 2025). Diferentemente de *firewalls*, que atuam como barreiras de acesso, os IDS são especializados em detectar padrões

de invasão e alertar as equipes de segurança. Recentemente, como demonstram Houda *et al.* (2022), a integração de algoritmos de Aprendizado Profundo (DL) tem aprimorado a eficácia dos IDS em ambientes IoT, permitindo que aprendam automaticamente os padrões de ataques e melhorem as taxas de detecção.

Os IDS *open source* consolidaram-se como alternativa viável para organizações que demandam soluções de segurança acessíveis e altamente adaptáveis. Ao contrário das soluções proprietárias, que frequentemente apresentam custos proibitivos e arquiteturas fechadas, essas ferramentas oferecem flexibilidade para modificações tanto no módulo de captura de pacotes quanto nos mecanismos de detecção, particularmente nos algoritmos de correspondência de padrões (Waleed *et al.*, 2022). Essa adaptabilidade permite ajustes precisos conforme as necessidades específicas de cada ambiente de rede.

Dentre as soluções de IDS de código aberto disponíveis, destacam-se as ferramentas Snort, Suricata e Zeek como as mais consolidadas e amplamente adotadas para proteção de redes. Conforme estudo comparativo de Waleed *et al.* (2022), essas plataformas oferecem implementações robustas dos modos IDS e IPS, com exceção do Zeek, que opera exclusivamente como sistema de detecção. Snort é o sistema de prevenção de intrusões *open source* mais utilizado mundialmente, empregando regras personalizáveis para identificar atividades maliciosas (SNORT, n.d.). Suricata oferece detecção em tempo real com suporte a protocolos modernos e análise aprofundada de pacotes (SURICATA, n.d.). Já Zeek destaca-se pela geração de logs detalhados e análise personalizável de tráfego, com foco exclusivo em detecção (ZEEK, 2023).

A implementação de IDS em ambientes de IoT enfrenta desafios significativos devido às demandas únicas desses ecossistemas. Pesquisas de Altulaihan *et al.* (2024) demonstram que dispositivos de IoT requerem sistemas de detecção leves e eficientes, capazes de operar com recursos computacionais limitados. Contudo, o desenvolvimento de soluções adequadas encontra obstáculos, incluindo a necessidade de otimização rigorosa de recursos, onde recursos redundantes podem comprometer o desempenho, e a escassez de conjuntos de dados atualizados que representem adequadamente as ameaças cibernéticas atuais. Essa combinação de fatores torna crucial o desenvolvimento de abordagens inovadoras que equilibrem a precisão da detecção com o consumo eficiente de recursos.

Diante desses desafios, este trabalho tem como objetivo realizar uma análise comparativa das ferramentas Snort, Suricata e Zeek no contexto de redes IoT/Internet Industrial das Coisas (IIoT), a partir da interpretação do tráfego de rede de dispositivos IoT. Para isso, será

utilizado o dataset *Edge-IIoTset*, que reúne arquivos Packet Capture (PCAP) contendo tanto tráfego normal quanto diferentes classes de ataques, simulando tráfegos realísticos de operação em ambientes IoT. A execução dos experimentos será conduzida por meio do *framework* Dalton, que possibilita a análise *offline* de tráfego previamente capturado, garantindo padronização, isolamento e reprodutibilidade na execução dos IDS. Cada ferramenta é avaliada sobre os mesmos conjuntos de dados, permitindo uma comparação direta quanto à sua capacidade de identificar atividades maliciosas e ao comportamento dos alertas gerados.

A avaliação dos IDS concentra-se na análise conjunta da eficácia de detecção e do desempenho computacional, considerando métricas como quantidade de alertas, falsos positivos, verdadeiros positivos e falsos negativos, bem como o consumo de CPU e memória RAM. Essas métricas, obtidas a partir da execução das soluções em contêineres Docker, permitem avaliar de forma integrada o impacto da interpretação do tráfego IoT tanto na acurácia da detecção quanto na utilização de recursos computacionais.

1.1 Objetivos

1.1.1 Objetivo geral

Realizar uma avaliação comparativa e reprodutível das ferramentas *open-source* de detecção de intrusão Snort, Suricata e Zeek em tráfego IoT, utilizando o *framework* Dalton e o *dataset Edge-IIoTset*, com foco na eficácia de detecção baseada em assinaturas.

1.1.2 Objetivo específicos

1. Estabelecer o protocolo experimental reprodutível e realizar a curadoria do *dataset Edge-IIoTset*, definindo parâmetros de execução no *Dalton*, segmentando arquivos volumosos e catalogando metadados e *hashes* que servirá como referência experimental;
2. Implementar os ensaios experimentais de forma isolada para as ferramentas *Snort*, *Suricata* e *Zeek*, realizando a submissão padronizada de *jobs* por PCAP e a extração automatizada dos artefatos de saída (`json`, `csv` e logs de conexão);
3. Realizar uma análise qualitativa comparativa entre os IDS, identificando limitações semânticas e propondo recomendações técnicas para segurança em redes IoT.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, são apresentados os conceitos fundamentais para a compreensão desta pesquisa, iniciando pela análise das arquiteturas de redes IoT e suas vulnerabilidades na Seção 2.1, que abordará os modelos de três e cinco camadas e as abordagens baseadas em computação em nuvem e névoa. A Seção 2.2 explorará os sistemas de detecção e prevenção de intrusões, examinando os princípios dos Sistema de Prevenção de Intrusão (IPS) com suas variantes baseadas em Sistema de Prevenção de Intrusão Baseado em Host (HIPS) e Sistema de Prevenção de Intrusão Baseado em Rede (NIPS), bem como os métodos de detecção de intrusão, incluindo técnicas baseadas em assinatura, anomalia, abordagens híbridas, e implementações baseadas em host (HIDS) e rede (NIDS). As Seções 2.2.3.1, 2.2.3.2 e 2.2.3.3 detalharão respectivamente as ferramentas Snort, Suricata e Zeek, analisando suas arquiteturas e mecanismos de detecção.

2.1 Redes IoT: Arquitetura e Paradigmas Computacionais

A capacidade dos dispositivos IoT de coletar e comunicar dados os torna poderosos, mas também vulneráveis a ameaças cibernéticas, um desafio crítico para fabricantes e pesquisadores, considerando o crescimento exponencial de dispositivos conectados (Ahmad *et al.*, 2022). Essa vulnerabilidade é agravada pela própria natureza da IoT, que não consiste em uma única tecnologia, mas em um ecossistema complexo de sistemas interconectados. Essa natureza multifacetada inviabiliza a adoção de um modelo arquitetural único como referência para todas as implementações possíveis (Maschietto *et al.*, 2021).

Diante dessa complexidade e da ausência de consenso sobre um padrão universal, a comunidade científica tem proposto diversas abordagens arquiteturais (Sethi; Sarangi, 2017). Entre os modelos mais relevantes de Internet das Coisas - Arquitetura (IoT-A), destacam-se dois paradigmas complementares: a Arquitetura de Camadas, que organiza funcionalidades de forma hierárquica, e a Arquitetura baseada em Nuvem e Névoa, que otimiza o processamento distribuído entre borda e núcleo da rede.

2.1.1 Arquitetura de Três e Cinco Camadas

A arquitetura de três camadas: percepção, rede e aplicação, ilustarada na Figura 1, consolidou-se como modelo básico da IoT, sendo simples de implementar e adequada para soluções menos complexas (Kumar; Mallick, 2018). A camada de percepção atua como a

interface física, equipada com sensores que permitem a detecção e a coleta de dados sobre o ambiente ao redor. Essa camada é capaz de monitorar diversos parâmetros físicos e reconhecer outros dispositivos inteligentes presentes. Por sua vez, a camada de rede tem a função de estabelecer conexões com outros dispositivos inteligentes, redes e servidores. Ela também desempenha um papel crucial na transmissão e no processamento das informações coletadas pelos sensores. Finalmente, a camada de aplicação é responsável por oferecer serviços específicos aos usuários, definindo uma variedade de aplicativos onde a Internet das Coisas (IoT) pode ser implementada, como em residências inteligentes, cidades conectadas e soluções de saúde inovadoras (Sethi; Sarangi, 2017). Entretanto, sua estrutura minimalista limita aplicações que demandam integração de múltiplas tecnologias ou exigem maior granularidade operacional, como sistemas de transporte inteligente ou ambientes industriais avançados (Kumar; Mallick, 2018).

Para suprir essas lacunas, a arquitetura de cinco camadas, ilustrada na Figura 1, amplia o modelo original ao incorporar camadas dedicadas ao processamento de dados e à gestão de negócios, oferecendo suporte a análises em tempo real e à tomada de decisões estratégicas (Kumar; Mallick, 2018). A camada de transporte é responsável por transferir os dados coletados pelos sensores da camada de percepção para a camada de processamento e vice-versa, utilizando diversas redes, como *wireless*, 3G, LAN, *Bluetooth*, RFID e NFC. A camada de processamento, frequentemente referida como middleware, tem a função de armazenar, analisar e processar grandes volumes de dados recebidos da camada de transporte. Além disso, ela pode gerenciar e oferecer uma ampla gama de serviços para as camadas inferiores. Para isso, utiliza várias tecnologias, incluindo bancos de dados, computação em nuvem e módulos de processamento de big data. Por outro lado, a camada de negócios supervisiona todo o sistema da IoT, abrangendo aplicações, modelos de negócio, rentabilidade e a privacidade dos usuários (Sethi; Sarangi, 2017). Essa evolução reflete a necessidade de equilibrar simplicidade e robustez em cenários onde a IoT atua como espinha dorsal de sistemas críticos, como mobilidade urbana e automação industrial, garantindo escalabilidade sem comprometer a funcionalidade essencial (Kumar; Mallick, 2018).

2.1.2 Arquiteturas Baseadas em Nuvem e Névoa

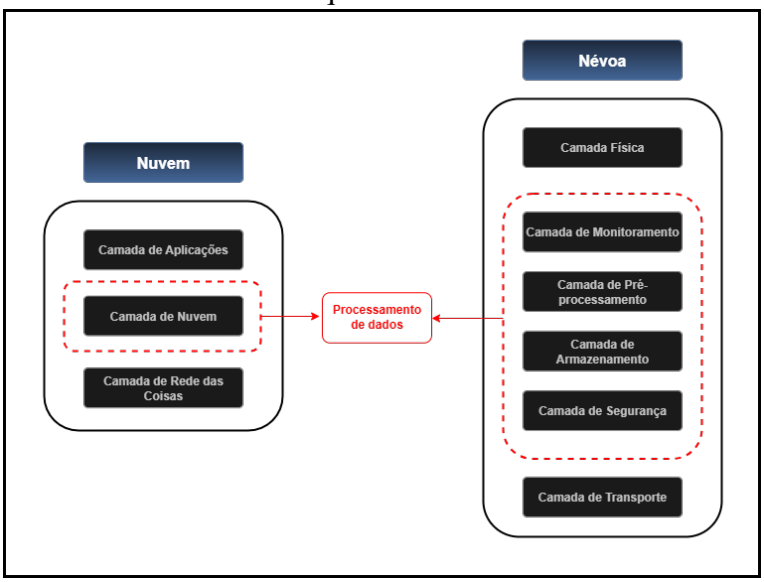
Como observado em algumas arquiteturas de sistemas, o processamento dos dados gerados pelos dispositivos IoT é realizado de forma centralizada por infraestruturas de computação em nuvem. Essa organização da rede, ilustrada na Figura 2, posiciona a nuvem como

Figura 1 – Arquiteturas IoT de 3 e 5 camadas.

Fonte: Elaborado pelo autor.

elemento central, com as aplicações em nível superior e a rede de objetos inteligentes em camadas inferiores (Sethi; Sarangi, 2017). A computação em nuvem é priorizada por sua flexibilidade e escalabilidade, oferecendo serviços de infraestrutura, plataforma, software e armazenamento, permitindo que desenvolvedores utilizem ferramentas de armazenamento, software, mineração de dados, aprendizado de máquina e visualização (Sousa *et al.*, 2009).

Figura 2 – Processamento de dados em arquiteturas de IoT baseadas em nuvem e névoa.



Fonte: Elaborado pelo autor, adaptado de Kumar e Mallick (2018).

Outro paradigma para lidar com o volume massivo de dados gerados por dispositivos IoT é a computação em névoa (*fog computing*), que surge como uma abordagem inovadora capaz

de reduzir a latência em sistemas de tempo real, minimizar o tráfego entre a borda e o núcleo da rede, e descentralizar o processamento da nuvem por meio de uma hierarquia de análise (Zyrianoff *et al.*, 2019). Diferentemente da nuvem, sua arquitetura processa dados parcialmente nos próprios sensores e gateways (Figura 2), seguindo um modelo de seis camadas: física (coleta de dados), monitoramento (recursos e serviços), pré-processamento (filtragem), armazenamento (formato/distribuição), segurança (privacidade) e transporte (protocolos) (Kumar; Mallick, 2018). Essa estrutura avançada permite análise localizada na borda, otimizando eficiência operacional em cenários críticos, como Internet Industrial das Coisas (IIoT), onde a resposta rápida e a segurança são prioritárias.

2.2 Sistemas de Detecção e Prevenção de Intrusões

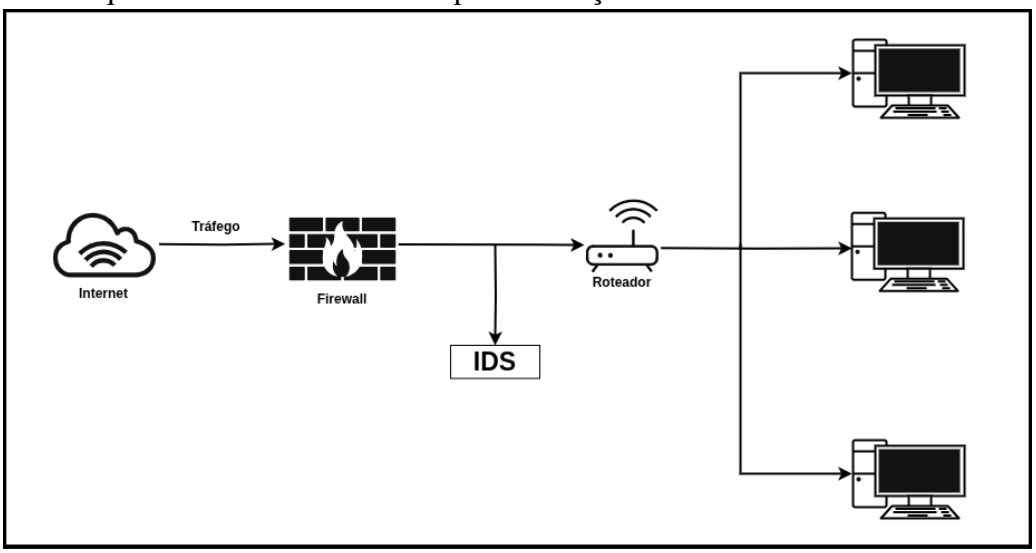
A proteção de dados é um dilema presente desde o surgimento dos primeiros computadores pessoais. Em 1987, Dorothy Denning, com seu artigo *An Intrusion Detection Model*, forneceu uma estrutura metodológica e deu início aos estudos sobre detecção de intrusão (Chakraborty, 2013). Os Sistema de Detecção de Intrusão (IDS) e IPS desempenham um papel crucial na segurança de redes. Atuando de forma complementar, protegem redes pessoais e corporativas contra ameaças externas. Apesar das semelhanças em nomenclatura e funcionamento, apresentam diferenças fundamentais em sua operação.

2.2.1 IDS

A detecção de intrusão consiste no monitoramento de atividades em sistemas ou redes para identificar possíveis ameaças, como malware, ataques DoS/DDoS, acessos não autorizados e tentativas de escalonamento de privilégios. O desafio está em diferenciar comportamentos maliciosos de ações legítimas, que podem ocorrer por erro do usuário. Nesse contexto, o IDS automatiza esse processo, facilitando a identificação de ataques (Ozkan-Okay *et al.*, 2021). Afinal, um IDS tem como função monitorar a rede e identificar atividades suspeitas, maliciosas ou violações de políticas por meio do tráfego, permitindo que os administradores mantenham um acompanhamento constante das ameaças atuais (Abdulganiyu *et al.*, 2023). A Figura 3 ilustra como um IDS é usado para monitorar o tráfego de rede em busca de atividades suspeitas e gerar alertas quando tal atividade é descoberta pelo sistema.

Antes do desenvolvimento dos IDS modernos, a detecção de intrusão consistia em

Figura 3 – Arquitetura de Rede com IDS para Detecção de Intrusões.



Fonte: Elaborado pelo autor.

uma busca manual por anomalias. Com a evolução do poder de processamento, tornou-se possível não apenas procurar padrões de ataque após a ocorrência, mas também realizar monitoramento contínuo, disparando alertas sempre que uma intrusão fosse detectada (Chakraborty, 2013).

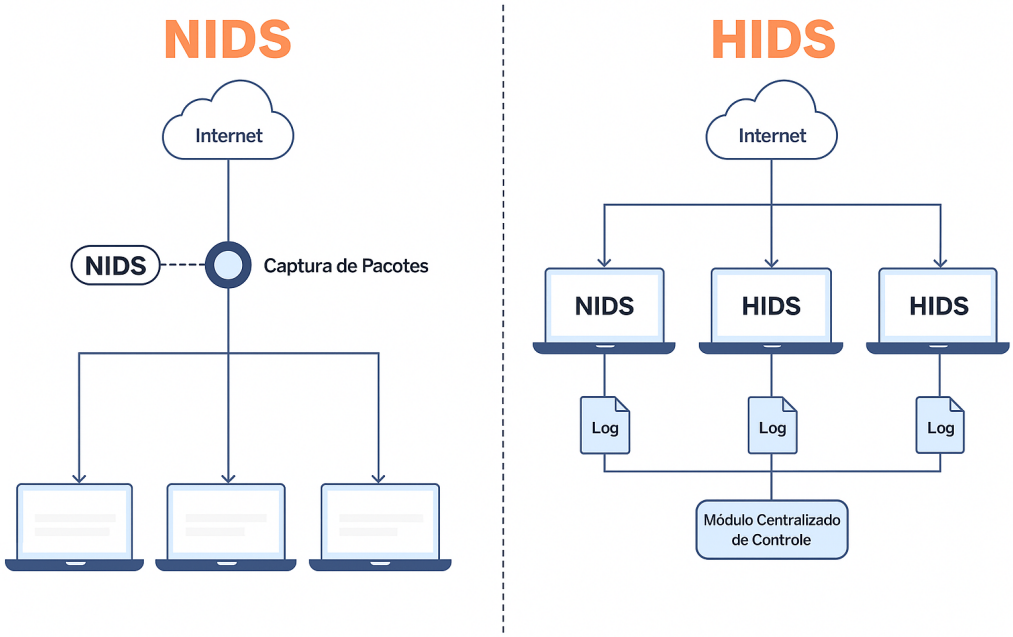
Diversas tecnologias IDS são utilizadas, cada uma oferecendo diferentes capacidades de coleta, gravação, detecção e prevenção de informações (Ozkan-Okay *et al.*, 2021). Dessa forma, torna-se crucial analisar o ambiente em que o IDS será aplicado, a fim de escolher a implementação mais adequada e evitar falhas na identificação de possíveis atividades hostis (Visky *et al.*, 2024). Segundo Abbas *et al.* (2023), os Sistemas de Detecção de Intrusão são classificados em dois tipos gerais: baseados em assinatura e baseados em heurística. Aqueles que operam em uma única estação são conhecidos como Sistema de Detecção de Intrusão Baseado em Host (HIDS), enquanto os que atuam como dispositivos autônomos na rede são chamados de Sistema de Detecção de Intrusão Baseado em Rede (NIDS). Na Figura 4 é ilustrado o funcionamento desses dois tipos.

Ambas implementações utilizam três tipos diferentes de detecção, sendo elas: Detecção de Uso Indevido (Misuse Detection), Detecção de Anomalias (Anomaly Detection) e Detecção Híbrida (Hybrid Detection) (Abbas *et al.*, 2023).

2.2.1.1 Sistema de detecção baseado em assinatura

A detecção por uso indevido, também chamada de detecção por assinatura, identifica padrões conhecidos de ataques na rede ou em um host. Cada ameaça possui uma assinatura específica, como um *payload*, um endereço IP de origem ou características nos cabeçalhos dos

Figura 4 – Tipos de IDS: monitoramento do tráfego de rede (NIDS) e análise de atividades no host (HIDS).



Fonte: Traduzido e adaptado de CyberHub (2023). Disponível em: https://cyberhub.sa/posts/5259. Acesso em: 30 maio 2025.

pacotes. Quando o IDS reconhece uma dessas assinaturas registradas, gera um alerta. Embora seja eficaz na identificação de ameaças conhecidas, não detecta ataques novos ou desconhecidos, como os de dia zero (Barika *et al.*, 2009).

Embora apresente limitações, a detecção baseada em assinaturas permanece como a metodologia mais amplamente adotada em IDS. Isso se deve à sua eficácia comprovada na identificação de ataques conhecidos, baixa complexidade computacional (ideal para operação online em tempo real) e implementação direta via regras SNORT/Suricata (Otoum; Nayak, 2021).

2.2.1.2 Sistema de detecção baseado em anomalia

A detecção por anomalia se destaca pela capacidade de reconhecer padrões, aprender com a experiência e detectar comportamentos anormais (Martins *et al.*, 2022). Ela define um comportamento normal da rede ou do host e sinaliza qualquer desvio como possível ameaça. Por exemplo, se ocorrer acesso incomum durante um horário atípico, o sistema emite um alerta. Apesar de eficiente na identificação de ataques inéditos, essa técnica sofre com dificuldades para definir precisamente o que é considerado normal, o que pode gerar muitos falsos positivos (Altulaihan *et al.*, 2024).

No entanto, a maioria das abordagens não atendem aos requisitos exigidos, sendo frequentemente avaliadas com base em conjuntos de dados obsoletos e não representativos do ambiente de trabalho. A construção de um modelo baseado em anomalias é realizado a partir de dados anteriores proeminentes de atividades da rede ou do sistema, partindo do pressuposto de que qualquer ataque, interpretado como uma anomalia, terá uma dinâmica diferente, sinalizando quaisquer discrepâncias como suspeitas (Martins *et al.*, 2022).

2.2.1.3 Sistema de detecção hibrido

Já a detecção híbrida combina ambas as abordagens, unindo a precisão da detecção por assinatura com a capacidade de identificar ameaças desconhecidas da detecção por anomalia, reduzindo falsos positivos e ampliando a eficácia contra novos ataques (Kurnala *et al.*, 2023). Na arquitetura híbrida, a análise de anomalias detecta comportamentos não mapeados, enquanto o método de assinaturas intercepta ameaças conhecidas, incluindo tentativas de adulteração dos parâmetros de treinamento do sistema para mascarar atividades maliciosas (Patcha; Park, 2007).

Otoum; Nayak (2021) destaca que a detecção híbrida proporciona maior segurança e eficácia na monitoração. Isso porque um IDS híbrido combina múltiplos algoritmos de *machine learning*, tais como: árvore de decisão C5, Máquinas de Vetores de Suporte (SVM), OC-SVM e k-Vizinhos mais próximos (k-NN). Adicionalmente, emprega técnicas de *deep learning*, como Redes Neurais Convolucionais (CNN) e Redes Neurais Recorrentes (RNN) para aprimorar a detecção de ataques, elevando significativamente sua eficiência. Embora o sistema híbrido ainda possa ignorar certos tipos de ataques, sua taxa reduzida de alarmes falsos aumenta a probabilidade de examinar a maioria dos alertas.

2.2.1.4 Sistema de Detecção de Intrusão Baseado em Host (HIDS)

Um IDS baseado em host tem como principal função monitorar o tráfego do servidor onde está instalado, registrar arquivos e transações com base em um banco de dados de assinaturas de ataques personalizado para esse servidor e responder prontamente na detecção de ameaças (CELAL BAYAR UNIVERSITY, 2022). Parâmetros como uso de memória, carga da CPU, tráfego de rede, processos e ações do usuário, são utilizados na detecção de possíveis ataques nos computadores em que o NIDS está instalado, fornecendo um nível mais profundo e localizado de análise de segurança. (Visky *et al.*, 2024)

Como explicado por CELAL BAYAR UNIVERSITY (2022), um IDS baseado em

host permite o monitoramento e resposta em tempo real, além de, em alguns casos, acompanhar atividades em portas e bloquear o acesso a portas privadas, reforçando a segurança da rede. Esses sistemas analisam os registros de tráfego do servidor utilizando bancos de dados de assinaturas e alertam os administradores sempre que identificam comportamentos suspeitos ou ataques imprevistos. A maioria dos HIDS utiliza agentes instalados diretamente nos hosts de interesse, responsáveis por monitorar as atividades locais. Esses agentes enviam informações para servidores de gerenciamento, que frequentemente integram bancos de dados para análise. O acompanhamento e a configuração são feitos por meio de consoles específicos. Em alguns casos, dispositivos dedicados executam o software do agente sem a necessidade de instalação nos *hosts*, monitorando o tráfego de máquinas específicas. Apesar de sua função ser semelhante à dos HIDS, tecnicamente esses dispositivos podem ser classificados como IDS baseados em rede (Ozkan-Okay *et al.*, 2021).

Apesar de apresentarem maior confiabilidade, esses sistemas ainda possuem algumas limitações e não são capazes de atender completamente todas as demandas do ambiente em que estão instalados. Uma dessas dificuldades é a falta de compatibilidade entre sistemas operacionais, o que exige que os HIDS sejam implementados especificamente para o sistema operacional em que serão utilizados (CELAL BAYAR UNIVERSITY, 2022).

2.2.1.5 Sistema de Detecção de Intrusão Baseado em Rede (NIDS)

Um NIDS é um sistema projetado para monitorar e analisar o tráfego dos dispositivos de rede e os protocolos (rede, aplicativo, transporte, etc.) que foram usados, a fim de detectar anomalias, intrusões ou violações (Nguyen; Kashef, 2023). Eles costumam operar como um dispositivo autônomo que monitora o tráfego na rede para detectar ataques (Abbas *et al.*, 2023).

NIDS recebem como entrada os dados coletados por uma unidade de captura, que extrai informações dos cabeçalhos dos pacotes (sem acessar o conteúdo das mensagens) e organiza esses dados em fluxos unidirecionais. As informações capturadas são armazenadas em arquivos simples para posterior análise em lote. Antes de serem processados pelo módulo de detecção de anomalias, realiza-se uma etapa de filtragem, na qual o analista remove tráfego irrelevante, como dados provenientes de fontes confiáveis ou padrões de comportamento previamente reconhecidos como não maliciosos (Raghunath; Mahadeo, 2008).

IDS baseados em rede oferecem uma capacidade robusta de detecção, combinando frequentemente técnicas baseadas em assinatura e em anomalia para realizar análises mais preci-

sas e aumentar a taxa de detecção. Nesse modelo, o método por anomalia avalia comportamentos suspeitos, enquanto as requisições e respostas são verificadas e comparadas com assinaturas de ataques conhecidos, tornando a aplicação desses métodos de forma integrada e complementar (Ozkan-Okay *et al.*, 2021).

Esse tipo de sistema opera em tempo real, capturando e analisando pacotes de dados à medida que trafegam pela rede. Essa capacidade permite a identificação imediata de atividades suspeitas, possibilitando respostas rápidas a potenciais ameaças. Contudo, os NIDS enfrentam desafios significativos. Em ambientes de alta velocidade, podem ocorrer perdas de pacotes devido à sobrecarga de tráfego, comprometendo a eficácia da detecção. Além disso, a análise de tráfego criptografado representa uma limitação, pois os NIDS geralmente não conseguem inspecionar o conteúdo desses pacotes, o que pode permitir que atividades maliciosas passem despercebidas. Outro desafio é a suscetibilidade a falsos positivos e negativos, que podem sobrecarregar os administradores de segurança com alertas irrelevantes ou deixar de identificar ameaças reais. Portanto, embora os NIDS sejam ferramentas valiosas para a segurança de redes, é essencial estar ciente de suas limitações e complementar sua utilização com outras medidas de segurança (Goldschmidt; Chudá, 2025).

2.2.2 Sistema de Prevenção de Intrusão (IPS)

O IPS possui como função principal detectar e bloquear atividades maliciosas na rede. Ele registra em logs todos os pacotes suspeitos identificados, bloqueia automaticamente o tráfego considerado perigoso e notifica o administrador da rede (Syafril *et al.*, 2024). Conforme explica Sheeraz *et al.* (2024), na segurança de redes, os sistemas IPS são geralmente preferidos aos IDS devido à sua capacidade de bloquear ativamente o tráfego malicioso.

Os IPS são soluções que podem ser implementadas como software, hardware ou serviços em nuvem, atuando sempre em linha na rede para bloquear ameaças em tempo real. Esses dispositivos não exigem grandes reconfigurações para serem implantados e realizam inspeção profunda do tráfego, capazes de identificar ameaças como vírus, worms, ataques a sistemas operacionais, vulnerabilidades em aplicações web, spyware, phishing, spam e até tráfego de aplicações não autorizadas, como redes P2P (Moraes, 2010). Os IPS podem ser classificados de acordo com a técnica de detecção utilizada e o local de implantação. Quanto à detecção, existem quatro abordagens principais: baseado em assinatura, baseado em anomalia, baseado em análise de protocolo (*stateful*) e híbrido (Gupta *et al.*, 2023). Dependendo da arquitetura

projetada, o IPS pode ser implementado no sistema e configurado de forma adequada. Enquanto o HIPS atua na segurança do endpoint, o NIPS é implementado na camada de rede.(Sawant, 2018)

2.2.2.1 Sistema de Prevenção de Intrusão Baseado em Host (HIPS)

O HIPS é uma solução instalada diretamente em um endpoint, como um notebook ou servidor, que monitora e protege exclusivamente o tráfego para esse dispositivo. Esse tipo de IPS utiliza agentes específicos para cada *host*, com regras e assinaturas atribuídas localmente, enquanto um servidor externo centraliza o gerenciamento dos *hosts* monitorados (Sawant, 2018).

Além de alertar sobre atividades suspeitas, o HIPS é capaz de bloquear ameaças originadas de nós comprometidos, como a propagação de *ransomware*. Normalmente, é utilizado em conjunto com o IPS baseado em rede (NIPS) para reforçar a segurança de ativos críticos. Ferramentas como OSSEC, *Trend Micro Deep Security Manager*, *Opensource* Tripwire e Wazuh são exemplos amplamente utilizados, e suas características foram analisadas e comparadas por Sawant em seu estudo, juntamente com o Snort (Gupta *et al.*, 2023).

2.2.2.2 Sistema de Prevenção de Intrusão Baseado em Rede (NIPS)

O NIPS monitora o tráfego de entrada e saída de todos os dispositivos conectados, inspecionando pacotes em busca de atividades suspeitas e bloqueando fluxos maliciosos (Abbas *et al.*, 2023). Esses sistemas são posicionados em pontos estratégicos da infraestrutura, normalmente logo após o *firewall* no perímetro, para detectar e impedir que tráfego malicioso avance, mas também podem ser implantados internamente para proteger ativos críticos, como data centers ou servidores sensíveis (Sawant, 2018).

O NIPS é capaz de detectar e bloquear tanto ataques externos quanto internos, protegendo toda a rede (Gupta *et al.*, 2023). Exemplos comuns de soluções NIPS incluem Snort, Suricata, Cisco IPS, Cisco IOS e IBM NIPS (Sawant, 2018).

2.2.3 Ferramentas IDS/IPS Open Source

2.2.3.1 Snort

Desenvolvido por Martin Roesch em 1998, Snort é um NIPS e um NIDS de código aberto ampalmente utilizado (Chakraborty, 2013). Snort captura e monitora em tempo real o

tráfego da rede analisando os pacotes recebidos para detectar ataques com base em assinaturas e quando uma regra é atendida ele pode registrar a ocorrência e exibir estatísticas diretamente no console (Waleed *et al.*, 2022). Dentre os ataques detectados pelo Snort estão DoS, DDoS, varreduras de porta, varreduras Nmap, sondas SBM, ataques CGI e consultas NetBIOS (Ahmad *et al.*, 2022).

O Snort é composto por diferentes módulos que desempenham funções específicas. Dentre seus principais componentes estão: o módulo de captura de pacotes, o decodificador, os pré-processadores, o mecanismo de detecção, o sistema de registro e alerta, e o módulo de saída (Ahmad *et al.*, 2022). Sua arquitetura modular permite monitorar o tráfego de rede de forma eficiente, embora uma limitação comum resida nas regras criadas, que podem ser redundantes. Além disso, o Snort não oferece detecção baseada em anomalias nativamente, o que pode limitar sua capacidade de identificar ataques desconhecidos.

Ainda assim, ele pode lidar com protocolos industriais, como o DNP3, desde que sejam utilizadas regras adequadas (Visky *et al.*, 2024). Por padrão, o Snort utiliza o *Libpcap* (biblioteca de captura de pacotes) para coletar pacotes de rede das várias interfaces em que está implementado (Ahmad *et al.*, 2022). Em seguida, utiliza um decodificador de pacotes para analisar os cabeçalhos.

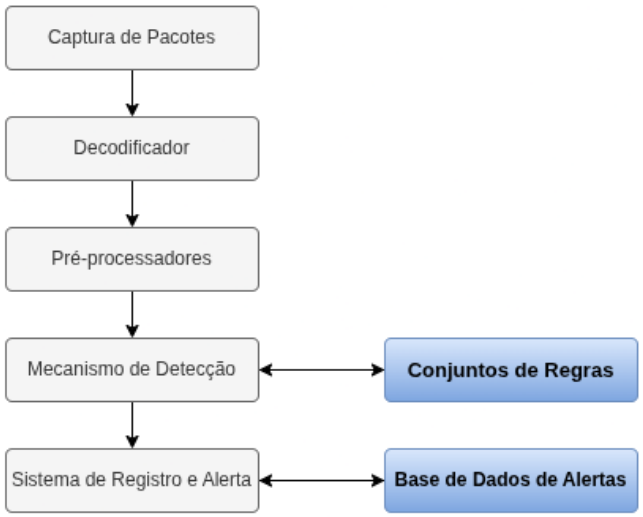
A normalização dos pacotes é realizada por um pré-processador, que converte o tráfego para um formato compreensível pelo mecanismo de detecção, possibilitando que sejam realizadas remontagens de fluxos Transmission Control Protocol (TCP), Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), Secure Shell (SSH), entre outros (Waleed *et al.*, 2022). Posteriormente, o mecanismo de detecção analisa o tráfego aplicando regras específicas para identificar pacotes maliciosos.

O Snort opera com base na detecção por assinatura (uso indevido) e, por padrão, não realiza detecção por anomalias. Quando configurado como IDS, apenas emite alertas ao identificar ameaças; já no modo IPS, além de detectar, também bloqueia pacotes suspeitos. Com base nessa saída, o Snort gera os alertas correspondentes (Ahmad *et al.*, 2022). Todo esse fluxo é representado na Figura 5.

Ele está evoluindo gradualmente. A comunidade mundial (bastante ativa) revisa, testa e oferece atualizações continuamente para o código fonte do Snort. Ela também aceita revisões e bancos de dados de conhecimento de especialistas em segurança ao redor do mundo

para implementar alterações. Outro fator para sua popularidade é o suporte a vários sistemas operacionais, como Unix, Linux, FreeBSD, macOS e Windows (Waleed *et al.*, 2022).

Figura 5 – Funcionamento Componentes Snort.



Fonte: Tradução e redesenho do autor com base em (Munir *et al.*, 2016).

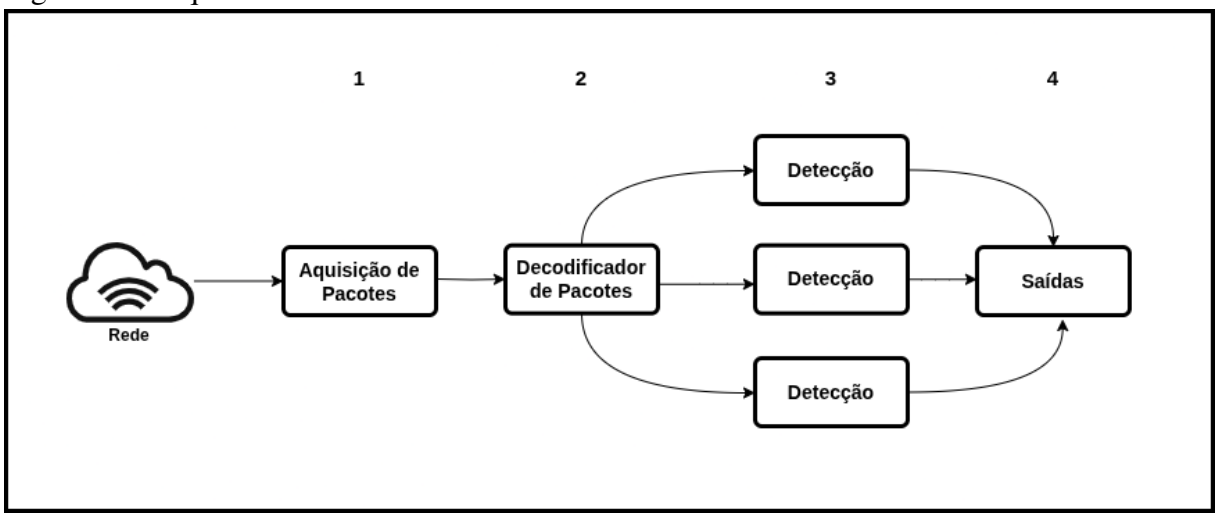
2.2.3.2 Suricata

Desenvolvido pela *Open Information Security Foundation* (OISF) e lançado em 2010, o Suricata é uma ferramenta IDS e IPS de código aberto. Sua criação atendeu, em parte, à demanda por um IDS com suporte a *multithreading*, uma limitação presente no Snort. Por esse motivo, o Suricata incorpora diversas funcionalidades semelhantes às do Snort, mas também introduz recursos que não estavam presentes em seu antecessor (Hoover, 2022). Projetado como um sistema multithread que utiliza múltiplos núcleos, proporcionando desempenho superior em máquinas multinúcleo com conjuntos de regras otimizados (Maliki *et al.*, 2024).

Em termos de detecção o Suricata tanto é utilizado com base em assinaturas e anomalisa, monitorando o comportamento do tráfego da rede e emitindo alertas caso atividades suspeitas sejam identificadas. Utiliza o mesmo formato empregado pelo Snort para declaração de regras e também possui algoritmos de detecção semelhantes (Waleed *et al.*, 2022). Conforme demonstra Visky *et al.* (2024), ele é segue a mesma arquitetura do Snort, com um módulo de captura, decodificação e detecção. No módulo de captura, os dados são coletados das interfaces de rede. No módulo de decodificação, os pacotes são decodificados para suportar a estrutura Suricata. No módulo de detecção, os pacotes são considerados normais ou de ataque com

base em regras, assinaturas e anomalias predefinidas (Ahmad *et al.*, 2022). Essa arquitetura é exemplificada na Figura 6.

Figura 6 – Arquitetura Multithread Suricata.



Fonte: Tradução e redesenho do autor com base em (Hoover, 2022).

O Suricata pode atuar como um sistema de detecção e prevenção de intrusões em rede (NIDS/NIPS), além de funcionar como monitor de segurança e registrador de pacotes no formato Packet Capture (PCAP). Ele conta com um mecanismo de fluxo escalável, suporte à análise de fluxos TCP e capacidade de desfragmentação de pacotes IP. É capaz de interpretar protocolos tanto na camada de enlace quanto na camada de aplicação, e possui um analisador HTTP com estado, capaz de registrar transações e extrair arquivos da comunicação. Seu motor de detecção é altamente eficiente, flexível e pode ser adaptado para atender a requisitos específicos. Um dos seus diferenciais mais relevantes é a compatibilidade com *multithreading*. A arquitetura de execução em múltiplos *threads* é totalmente personalizável, permitindo desde a operação em um único núcleo até o uso de diversas threads simultâneas. Além disso, é possível configurar a afinidade de CPU, permitindo que o Suricata atribua *threads* a núcleos específicos ou utilize todos os disponíveis no sistema (Hoover, 2022).

Apesar de suas funcionalidades avançadas, o Suricata apresenta algumas limitações que podem impactar sua adoção em determinados ambientes. Um dos principais desafios é o elevado consumo de CPU, especialmente em redes de alto tráfego, o que pode exigir hardware mais robusto. Além disso, as informações sobre atualizações nem sempre são amplamente divulgadas ou documentadas, o que pode dificultar a manutenção do sistema. Outro ponto a ser considerado é que a comunidade de suporte do Suricata é menos ampla e ativa quando comparada à do Snort, o que pode limitar o acesso rápido a soluções e boas práticas. Por fim, o processo de

instalação e configuração pode ser complexo para usuários iniciantes, exigindo conhecimento técnico mais aprofundado (Ahmad *et al.*, 2022).

2.2.3.3 Zeek

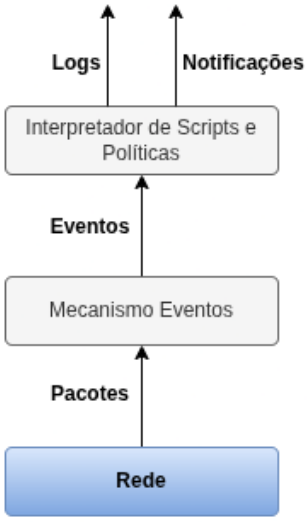
Lançado por Vern Paxson em 1995 no Laboratório Nacional Lawrence Berkeley, Zeek é um analisador passivo de tráfego de rede de código aberto. Por muitas vezes é utilizado como um monitor de segurança de rede (NSM) visando auxiliar em investigações de atividades suspeitas ou maliciosas (ZEEK, 2023). Essa capacidade faz com que ele também possa ser implementado como uma ferramenta de IDS que identifica anomalias no tráfego de rede (Ahmad *et al.*, 2022).

Como descrito em sua documentação (ZEEK, 2023), o Zeek não é um IDS baseado em assinaturas clássico. Ele é um NIDS baseado em anomalias, projetado para analisar tráfego denso de rede. Por esse fator, ele pode detectar ataques sem características conhecidas. Outro destaque é sua alta capacidade de personalização, pois e utiliza sua própria linguagem de script. A linguagem Zeek pode ser comparada a um "Python voltado para domínios específicos" (ou até mesmo ao Perl), pois, assim como essas linguagens, oferece um conjunto robusto de funcionalidades pré-definidas. Apesar disso, seu verdadeiro poder está na flexibilidade, permitindo os usuários a capacidade de criar scripts personalizados para realizar análises específicas. Na verdade, todas as análises padrão do Zeek, incluindo o registro de eventos, são implementadas via scripts, sem que haja lógica fixa incorporada diretamente no núcleo do sistema (Visky *et al.*, 2024).

O gerenciador do Zeek é composto por dois componentes, conforme ilustrado na Figura 7: um mecanismo de eventos e um interpretador de scripts de políticas. O mecanismo de eventos converte cada pacote capturado da rede em um evento, que é então repassado ao interpretador. Este, por sua vez, aplica as regras definidas nos scripts de política do Zeek a fim de analisar o comportamento da rede. Caso alguma atividade suspeita ou maliciosa seja identificada, o sistema pode gerar alertas em resposta. Um dos diferenciais do Zeek é sua capacidade de detectar ameaças com base em anomalias, o que o torna particularmente útil na identificação de ataques do tipo *zero-day*. Esse tipo de detecção não é oferecido por ferramentas como o Snort e o Suricata, que se baseiam exclusivamente em assinaturas conhecidas. Além disso, o Zeek proporciona uma análise detalhada do tráfego de rede, sendo uma ferramenta de código aberto eficaz, relativamente fácil de implementar e bastante flexível. Todavia, sua adoção mais ampla é

limitada pela quantidade reduzida de assinaturas/regras padrão disponíveis. Em contrapartida, o Snort e o Suricata contam com extensas bases de regras mantidas por comunidades ativas, o que contribui para sua maior popularidade em implementações práticas (Waleed *et al.*, 2022).

Figura 7 – Arquitetura Zeek.



Fonte: Elaborado pelo autor.

Apesar de suas capacidades avançadas, o Zeek apresenta algumas limitações que podem dificultar sua adoção em certos contextos. Uma das principais desvantagens é a falta de suporte para todos os sistemas operacionais, o que restringe sua aplicabilidade em ambientes heterogêneos. Além disso, sua utilização exige conhecimento técnico especializado, o que pode representar uma barreira para usuários iniciantes ou com pouca experiência na área de segurança de redes. Somam-se a isso desafios relacionados à usabilidade, como a ausência de interfaces gráficas intuitivas e um processo de instalação que pode ser considerado complexo (Ahmad *et al.*, 2022).

3 TRABALHOS RELACIONADOS

Ao revisar a literatura especializada, identificaram-se estudos relevantes que abordam diferentes abordagens e investigações sobre a detecção de intrusões em redes IoT utilizando ferramentas como Snort, Suricata e Zeek. Nesta seção, ocorrerá a apresentação das pesquisas apresentadas, destacando suas contribuições para o campo da segurança cibernética em IoT e como elas se relacionam com a análise comparativa proposta neste trabalho.

3.1 Comparative Study of Snort 3 and Suricata Intrusion Detection Systems

Em seu trabalho, Hoover (2022) realiza um estudo comparativo entre dois sistemas de detecção de intrusão em rede (NIDS) de código aberto: o recém-lançado `Snort 3` (com arquitetura multithread) e o `Suricata`. O objetivo principal é avaliar o desempenho e o comportamento de alerta (capacidade de detecção de ataques) de ambos os sistemas quando analisam o mesmo tráfego malicioso, utilizando as regras padrão recomendadas e projetadas especificamente para cada um. A solução apresentada consiste na configuração de ambos os NIDS (`Suricata v6.0.4` e `Snort 3 v3.1.6.0`) no mesmo ambiente virtual (`Ubuntu 18.04.6 LTS`) e na execução de testes utilizando arquivos `pcap` contendo tráfego malicioso gerado pela ferramenta `Pytbull`, incluindo um Ataque de Negação de Serviço (DoS) isolado e uma combinação de oito testes diferentes.

Para os testes foi utilizado um ambiente virtualizado com 4 núcleos de CPU, 4GB de RAM e 50GB de armazenamento. Ambos os NIDS foram instalados e configurados individualmente com suas configurações padrão recomendadas. As regras de cada sistema foram atualizadas para as versões mais recentes utilizando `PulledPork3` e `Suricata-Update`, respectivamente. O tráfego malicioso para análise foi obtido a partir de arquivos `pcap` públicos contendo dois cenários: um único ataque DoS e um traço combinado dos oito testes do `Pytbull`.

Para cada execução, foram coletados dados de utilização de recursos (CPU e memória) usando as ferramentas `nmon` e `htop`, e o número e os detalhes dos alertas gerados foram registrados e inspecionados. Um desafio metodológico significativo foi a incompatibilidade das regras entre os sistemas, levando à decisão de usar os *rulesets* nativos de cada NIDS e incluir a análise do comportamento de alerta como um foco adicional.

Os resultados revelaram que ambos os NIDS apresentaram desempenho muito similar na utilização da CPU, com média de aproximadamente 25% durante todos os testes. Isso ocorreu

porque cada sistema, apesar de suportar multithreading e estar configurado para isso, utilizou apenas um núcleo a 100%. Entretanto, o `Suricata` demonstrou ser mais intensivo em memória ($\sim$7,5% de RAM em ambos os testes) comparado ao `Snort 3` (4% no teste DoS e 5% no teste combinado). Na análise de comportamento de alerta, o `Suricata` detectou significativamente mais ataques do que o `Snort 3` usando seus respectivos *rulesets* padrão, gerando 2 vezes mais alertas no teste com o `pcap` menor (DoS) e 3,5 vezes mais alertas no teste com o `pcap` maior (8 testes). Esta disparidade é atribuída tanto às diferenças nos *rulesets* (ET Open para `Suricata` vs. Talos LightSPD para `Snort 3`) quanto às diferenças intrínsecas nos motores de detecção dos dois NIDS.

3.2 Which Open-Source IDS? Snort, Suricata or Zeek

(Waleed *et al.*, 2022) consiste em uma análise abrangente, inédita na literatura, que considera cenários típicos de Pequenas e Médias Empresas, variando módulos de captura de pacotes, algoritmos de detecção, tamanho de pacotes e regras. Busca-se orientar pesquisadores e profissionais na seleção e otimização de NIDPS conforme suas necessidades. O estudo aborda o desafio de processar tráfego de rede em alta velocidade em Sistemas de Detecção/Prevenção de Intrusão (NIDPS), problema que causa perda de pacotes e degradação de desempenho. O objetivo principal é avaliar comparativamente três soluções open-source, `Snort` (single/multi-threaded), `Suricata` e `Zeek`, utilizando parâmetros industriais como o consumo de recursos, taxa de perda de pacotes, latência, etc.

Para realizar seus testes, os autores adotaram dois métodos de avaliação: o impacto dos módulos e um benchmarking de desempenho. Os resultados revelaram que o `Suricata` superou todos os demais sistemas avaliados em todos os parâmetros testados, como consumo de recursos, processamento de pacotes, taxa de perda, *throughput* e latência, consolidando-se como a opção mais eficiente para implantação em pequenas e médias empresas. No entanto, foram identificadas limitações críticas no `Suricata`, com desempenho degradado em modo IPS e sob condições de estresse, indicando a necessidade de otimizações futuras. O `Zeek` apresentou restrições técnicas significativas, como incompatibilidade com `AF_Packet` (método de captura de pacotes) e ausência de modo IPS, o que compromete sua aplicabilidade comparativa. Já o `Snort 2.9.16` demonstrou desempenho variável conforme o algoritmo utilizado, embora ainda inferior ao `Suricata`. Todos os sistemas foram impactados pelo tamanho dos pacotes e das regras, mas o `Suricata` manteve sua vantagem mesmo com 42 mil regras. Assim, a conclusão

implícita é que o `Suricata` é a solução recomendada, embora ainda demande melhorias para cenários de prevenção (IPS) e alta carga, sendo a integração de algoritmos como o `Hyperscan` uma sugestão para trabalhos futuros.

3.3 A Review and Comparative Analysis of Intrusion Detection Systems for Edge Networks in IoT

Amrullah (2025) apresenta uma revisão abrangente e análise comparativa de IDS para redes em ambientes IoT, diante da crescente exposição a vulnerabilidades de segurança decorrentes da expansão da Internet das Coisas. O objetivo principal é examinar criticamente técnicas de IDS baseadas em assinatura, anomalia e métodos híbridos, com ênfase em aplicações de *machine learning* e *deep learning*, visando identificar soluções eficazes para dispositivos com restrições de recursos. A solução apresentada consiste na avaliação de arquiteturas de sistema, algoritmos (como `LSTM`, `CNN` e `Transformer`), conjuntos de dados e métricas de desempenho, para oferecer um recurso prático que auxilie pesquisadores e profissionais na seleção e implementação de IDS adaptados a cenários específicos de IoT.

O autor adotou uma abordagem sistemática de revisão de literatura, utilizando bases de dados reconhecidas e combinando palavras-chave estratégicas como *"Intrusion Detection System"*, *"IoT"*, *"Machine Learning"* e *"Security"*. Foram selecionadas exclusivamente publicações dos últimos cinco anos, focadas na aplicação de técnicas de aprendizado de máquina para IDS em IoT. O processo envolveu análise crítica de arquiteturas de sistema, algoritmos, conjuntos de dados e métricas de avaliação (precisão, *recall*, *F1-score*), com verificação rigorosa do rigor científico, representatividade dos dados e adequação das métricas. A síntese comparativa identificou tendências, lacunas e direcionamentos futuros por meio de avaliação qualitativa e comparação transversal das abordagens.

Finalmente, demonstrou com seus resultados o predomínio de técnicas de *deep learning* (como `LSTM` e `Transformers`) para detecção de intrusões em IoT, com destaque para modelos como o `LSTM-based L2D2` que apresentaram desempenho superior em ambientes complexos como Internet das Coisas Médicas. Verificou-se que a eficácia desses modelos está intrinsecamente vinculada à qualidade dos conjuntos de dados (ex: `CICIoT2024`), técnicas robustas de pré-processamento e tratamento de desequilíbrios de classes. Abordagens para ataques específicos, como DDoS, beneficiaram-se de estratégias inovadoras como janelas de tempo baseadas em entropia. Identificou-se ainda que a seleção ótima de características, métodos de

ensemble e integração com `blockchain` são críticos para segurança em ambientes heterogêneos, embora a implementação em dispositivos restritos permaneça um desafio. Conclui-se que soluções híbridas e adaptativas, aliadas a conjuntos de dados realistas, são essenciais para avanços futuros.

3.4 DDoS Attack Detection in Edge-IIoT using Ensemble Learning

(Laiq *et al.*, 2023) propõem a detecção de ataques DDoS (HTTP, ICMP, TCP e) em redes *Edge-IIoT*, utilizando o conjunto de dados específico *Edge-IIoT*. A justificativa para este estudo é a crescente vulnerabilidade das redes *Edge-IIoT* a ataques cibernéticos, especialmente DDoS, que podem causar indisponibilidade e prejuízos financeiros. Um dos objetivos específicos deste trabalho é aplicar técnicas de *ensemble learning*, empregando o algoritmo *XGBoost* e um classificador de *hard voting* que combina *Support Vector Machine* (SVM), *Decision Tree* (DT) e *Naive Bayes* (NB), abordagens consideradas inovadoras para este *dataset*.

A metodologia adotada envolveu um rigoroso pré-processamento do *dataset Edge-IIoT*, começando pela filtragem para reter apenas o tráfego normal e os quatro tipos específicos de ataques DDoS em foco. Foram removidas 15 colunas consideradas irrelevantes, seguidas de um embaralhamento dos dados para evitar sequências de tipos de ataque que pudessem enviesar o treinamento. Os valores categóricos foram convertidos em numéricos, e os dados foram divididos utilizando a função `train_test_split`, alocando 70% para treino e 30% para teste, com o estado aleatório fixado em 42 para garantir reprodutibilidade. As variáveis de entrada (X) incluíram todas as colunas, exceto as duas últimas (rótulos), enquanto a variável alvo (y) correspondeu especificamente ao rótulo de ataque, configurando um problema de classificação multiclasse. Este processo foi aplicado de forma idêntica para treinar e testar tanto o modelo *XGBoost* quanto o *ensemble* de *hard voting*.

Os resultados demonstraram uma diferença significativa de desempenho entre as duas abordagens de *ensemble learning* testadas. O modelo baseado em *XGBoost* alcançou um desempenho excepcional, atingindo 100% em todas as métricas de avaliação reportadas: Acurácia, Precisão, *Recall* e *F1-Score*. Em contraste, o classificador de *hard voting*, que agregou as previsões de SVM, DT e NB, obteve um desempenho consideravelmente inferior, com uma Acurácia de 88,77% e valores de Precisão, *Recall* e *F1-Score* consistentes em 88,8%. Isso representa uma vantagem de desempenho de aproximadamente 11% para o *XGBoost* em comparação com o *ensemble* de *hard voting*, destacando sua superioridade na tarefa de detecção

dos ataques DDoS no contexto específico do *Edge-IIoT* utilizando o *dataset* em questão.

3.5 Análise Comparativa

O Quadro 1 apresenta uma análise comparativa entre este trabalho e os estudos relacionados detalhados nas seções anteriores, visando posicionar a pesquisa proposta no contexto da literatura existente. Para isso, foram adotados cinco critérios principais: (i) comparação das ferramentas Snort, Suricata e Zeek; (ii) utilização de métricas de eficiência ou eficácia; (iii) avaliação de vulnerabilidades em redes IoT; (iv) emprego do dataset *Edge-IoTset*; e (v) realização de testes em ambiente virtual Linux.

Observa-se que todos os trabalhos analisados utilizam métricas de eficiência ou eficácia em suas avaliações. Porém, apenas o estudo de (Waleed *et al.*, 2022) e o trabalho atual comparam diretamente Snort, Suricata e Zeek. Na avaliação de vulnerabilidades IoT, destacam-se os trabalhos de (Amrullah, 2025), (Laiq *et al.*, 2023) e a presente pesquisa. Quanto ao dataset *Edge-IoTset*, apenas (Laiq *et al.*, 2023) e este trabalho o utilizam. Sobre os ambientes de teste, (Hoover, 2022), (Waleed *et al.*, 2022) e esta pesquisa realizam experimentos em ambientes virtualizados Linux.

Considerando esses aspectos, o diferencial do presente estudo reside na **combinação integral** desses cinco elementos, analisando ferramentas de detecção em ambientes IoT com métricas específicas, utilizando um dataset especializado e garantindo reprodutibilidade através de virtualização Linux.

Quadro 1 – Comparativo entre os trabalhos relacionados com o trabalho proposto.

Itens de Comparação	**Este trabalho**	**(Hoover, 2022)**	**(Waleed *et al.*, 2022)**	**(Amrullah, 2025)**	**(Laiq *et al.*, 2023)**
Comparação entre Snort, Suricata e Zeek	X		X		
Utilização de métricas de eficiência/eficácia	X	X	X	X	X
Avaliação de vulnerabilidades em redes IoT	X			X	X
Reprodutibilidade experimental	X	X	X		X

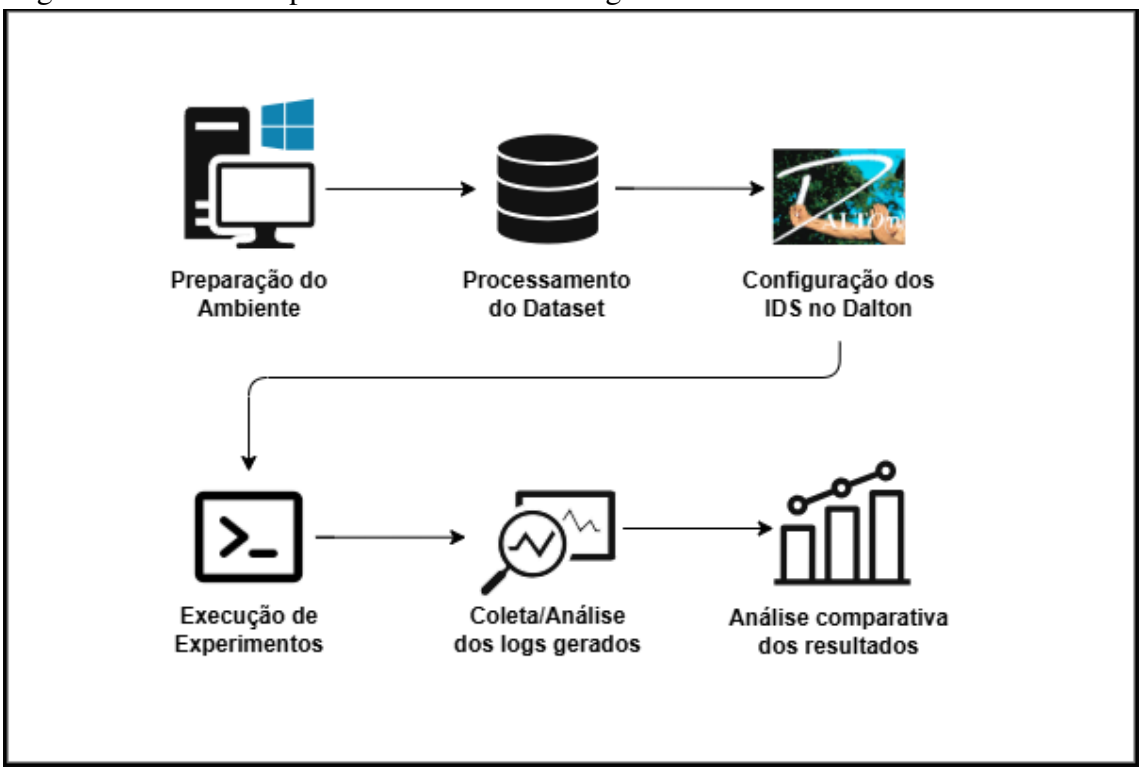
Fonte: elaborado pelo autor.

4 METODOLOGIA

Nesta seção, são apresentadas as etapas necessárias para a realização e avaliação deste trabalho. A metodologia proposta foi estruturada em quatro etapas principais, alinhadas aos objetivos específicos:

1. Preparação do ambiente experimental por meio da instalação e inicialização do *framework* Dalton via *Docker*;
2. Organização do *dataset* Edge-IIoTset, com a seleção de todos os PCAPs de ataque listados no conjunto de dados, incluindo a segmentação dos PCAPs com mais de 100MB em partes menores, com o auxílio do *Wireshark*;
3. Configuração das ferramentas Snort, Suricata e Zeek no ambiente do Dalton e execução dos experimentos mediante a criação de *jobs* isolados para cada *PCAP*, com posterior coleta dos logs gerados por cada IDS;
4. Análise comparativa dos resultados, a partir do processamento dos logs e do uso dos rótulos do *dataset*, para o cálculo das métricas de avaliação de detecção e das métricas de desempenho computacional, como consumo de CPU e memória RAM.

Figura 8 – Fluxo dos procedimentos metodológicos.



Fonte: Elaborado pelo autor.

4.1 Preparação do Ambiente Experimental

A etapa inicial da metodologia consiste na preparação do ambiente experimental, com a instalação e inicialização do *framework* Dalton em contêineres Docker, de modo a garantir um ambiente padronizado, reprodutível e isolado para a execução dos experimentos. Essa abordagem permite a orquestração controlada das ferramentas de detecção de intrusão analisadas, assegurando condições homogêneas durante os testes. O uso de contêineres possibilita ainda a reinicialização do ambiente a cada experimento, evitando interferências de execuções anteriores e preservando a consistência e a confiabilidade dos resultados obtidos.

4.1.1 Dalton

Lançado em novembro de 2017, Dalton (SECUREWORKS, 2023) é um sistema de código aberto desenvolvido pela *Secureworks*, projetado para executar capturas de pacotes de rede (PCAP) nos sistemas de detecção de intrusão (IDS) como Snort, Suricata e Zeek. Permite que analistas de segurança executem rapidamente os PCAPs contra um sensor IDS de sua escolha, utilizando conjuntos de regras predefinidos ou regras personalizadas.

Como destacado em (SOPHOS, 2020), o Dalton inclui uma interface *web* para o *Flowsynth*, outra ferramenta da *Secureworks* que simplifica a criação de definições de fluxo de rede personalizadas, que são então compiladas em pcaps para fins de teste. O autor também ressalta as ferramentas como contribuições valiosas para a comunidade de segurança. Elas são baseadas em soluções utilizadas internamente pela equipe de pesquisa da *Secureworks* (*Counter Threat Unit*) e foram liberadas para auxiliar analistas nas etapas críticas de validação de detecções, que muitas vezes são onerosas sem os recursos adequados.

Visando sintetizar e organizar as principais capacidades do Dalton no contexto de testes e validação de sistemas de detecção de intrusão, o Quadro 2 apresenta suas funcionalidades centrais e respectivos casos de uso, conforme descrito na documentação oficial do projeto e em análises técnicas especializadas.

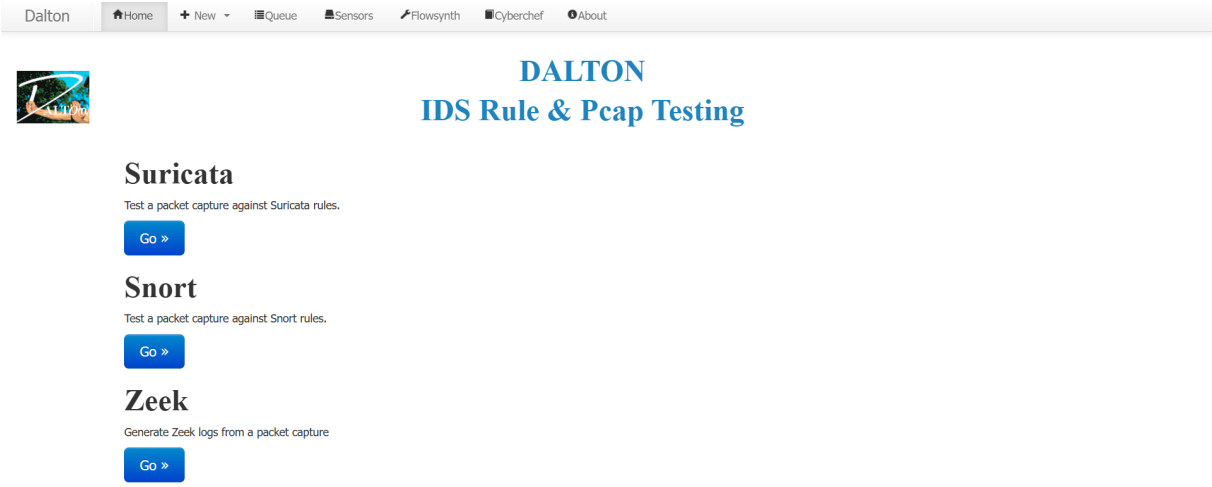
No quesito implantação, o Dalton fundamenta-se em uma arquitetura desacoplada e modular, composta por dois elementos centrais: o *Dalton Controller* e os *Dalton Agents*. O *Controller* atua como o ponto central de gerenciamento, provendo a interface *web* (apresentada na Figura 9) e a API RESTful, enquanto os *Agents* consistem em contêineres especializados que isolam as instâncias dos IDS.

Quadro 2 – Funcionalidades e principais casos de uso do Dalton IDS

Funcionalidade	**Descrição e casos de uso**
Teste de cobertura e desenvolvimento de assinaturas	Permite submeter arquivos *pcap* (pré-existentes ou gerados via Flowsynth) para verificar quais regras IDS são acionadas. Também possibilita o desenvolvimento, depuração e validação de assinaturas personalizadas, acelerando o ciclo de criação e *troubleshooting* de regras.
Teste de configurações e variáveis	Viabiliza a modificação dinâmica de arquivos de configuração do sensor (ex.: `suricata.yaml`) e variáveis de regras por job, permitindo avaliar o impacto de alterações como mudanças em redes internas ou externas (`EXTERNAL_NET`) sobre os alertas gerados.
Teste comparativo entre motores e versões	Suporta a execução do mesmo tráfego em múltiplos agentes, possibilitando comparações entre diferentes versões ou motores IDS, como Snort e Suricata, auxiliando na identificação de comportamentos específicos e regressões de detecção.
Integração e automação	Disponibiliza uma interface web aliada a uma API RESTful completa, permitindo a submissão de *jobs* e consulta de resultados de forma programática, facilitando a integração do Dalton em *pipelines* de automação e ambientes CI/CD.

Fonte: Elaborado pelo autor.

Figura 9 – Interface web do Dalton.



Fonte: Captura de tela da interface web do Dalton em ambiente local, feita pelo autor.

Esta separação, orquestrada via *Docker Compose*, permite que múltiplos sensores operem de forma simultânea e independente, garantindo que a análise de um PCAP em um motor (ex.: Snort) não sofra interferência de processos de outros motores (ex.: Zeek). Ao executar o *script* de inicialização (`./start-dalton.sh`), o sistema provisiona automaticamente o ambiente com todas as dependências e bibliotecas específicas de cada ferramenta, assegurando o determinismo e a reprodutibilidade dos experimentos — pilares essenciais para a validade científica desta pesquisa. Além disso, a natureza efêmera dos contêineres utilizados em cada *job* mitiga o risco de persistência de dados ou contaminação por resíduos de execuções anteriores, eliminando a complexidade de configurações manuais exaustivas no sistema operacional hospedeiro (SECUREWORKS, 2023).

4.2 Organização do Dataset *Edge-IIoTset*

Com a massiva quantidade de dados coletados por dispositivos da Internet das Coisas (IoT), empresas e pesquisadores têm investido no registro e análise dessas informações visando desenvolver aplicações para cidades inteligentes, automação industrial e monitoramento de saúde. Nesse contexto, com foco em cibersegurança, Ferrag *et al.* (2022) desenvolveram o conjunto de dados *Edge-IIoTset*, que consiste em uma seleção de dados abrangente que reflete ambientes IoT e IIoT realistas (Bukhari *et al.*, 2024). Conforme destacado por Laiq *et al.* (2023), para garantir uma segurança confiável e robusta, é imperativo a utilização de dados que correspondam às dificuldades encontradas por esses dispositivos em situações reais, permitindo a proposição de soluções eficazes.

O *Edge-IIoTset* engloba dez tipos de dispositivos IoT, detalhados na Tabela 1, incluindo sensores ultrassônicos (*HC-SR04*) para medição de distância em veículos autônomos, medidores de pH (*PH-4502C*) para monitoramento hídrico, detectores de nível de água, sensores capacitivos de umidade do solo e sensores de temperatura/umidade *DHT11*.

Tabela 1 – Dispositivos IoT utilizados no *Edge-IoTset*

Dispositivo	**Aplicação**	**Especificações**
Sensor Ultrassônico (HC-SR04)	Medição de distância	Alcance típico: 2 cm–400 cm
Sensor de Chama	Detecção de incêndio	Detecção de radiação infravermelha
Sensor de Batimentos Cardíacos	Monitoramento biométrico	Medição de frequência cardíaca
Receptor Infravermelho (IR)	Controle remoto / automação	Recepção de sinais IR
Sensor Modbus	Comunicação industrial	Protocolo Modbus RTU/TCP
Sensor de pH (PH-4502C)	Monitoramento hídrico	Faixa: 0–14 pH
Sensor de Umidade do Solo v1.2	Agricultura de precisão	Medição resistiva de umidade
Sensor de Som (LM393)	Monitoramento acústico	Detecção de níveis sonoros
Sensor de Temperatura e Umidade (DHT11)	Monitoramento ambiental	± 2 °C (temp.), $\pm 5\%$ (umid.)
Sensor de Nível de Água	Controle de reservatórios	Detecção de presença de líquido

Fonte: Adaptado de (Ferrag *et al.*, 2022).

Além da diversidade de hardware, o *dataset* incorpora **14 classes de ataques**, apresentadas na Tabela 2, que são categorizadas em cinco ameaças principais:

- **Man-in-the-middle**: ARP/DNS Spoofing com *Ettercap*;
- **DoS/DDoS**: Inundações TCP SYN (*hping3*), HTTP (*slowhttptest*) e Internet Control Message Protocol (ICMP);
- **Malware**: Ransomware com criptografia *OpenSSL* e backdoors via *Metasploit*;
- **Injeção**: XSS (*xsser*) e SQL Injection (*sqlmap*);

– **Coleta de informações**: Varredura de portas (*Nmap*) e fingerprinting de SO (*xprobe2*).

A distribuição dos dados revela um volume significativo de tráfego, contendo 11.223.940 registros normais (como leituras de sensores) e 9.728.708 registros de ataques, com destaque para inundações UDP e ICMP.

Tabela 2 – Estatísticas de tráfego normal e ataques no *Edge-IoTset*.

Tráfego IoT	**Classe**	**Registros**	**Total**
Normal	Normal	11.223.940	11.223.940
Ataque	Ataque de Backdoor	24.862	9.728.708
	Ataque DDoS_HTTP	229.022	
	Ataque DDoS_ICMP	2.914.354	
	Ataque DDoS_TCP	2.020.120	
	Ataque DDoS_UDP	3.201.626	
	Ataque de Fingerprinting	1.001	
	Ataque MITM	1.229	
	Ataque de Senha	1.053.385	
	Ataque de Varredura de Portas	22.564	
	Ataque de Ransomware	10.925	
	Ataque de Injeção SQL	51.203	
	Ataque de Upload	37.634	
	Ataque de Scanner de Vulnerabilidade	145.869	
	Ataque XSS	15.915	
Total			**20.952.648**

Fonte: Adaptado de (Ferrag *et al.*, 2022).

Originalmente, o conjunto foi composto por 1.176 *features* extraídas com as ferramentas *Zeek* e *TShark*, sendo posteriormente otimizado para 61 características críticas, incluindo protocolos IoT (Message Queuing Telemetry Transport (MQTT), *Modbus/TCP*), atributos de rede (*TCP flags*, *HTTP methods*) e metadados. O pré-processamento original empregou técnicas como *SMOTE* para balanceamento de classes minoritárias e *one-hot encoding*. Na avaliação proposta pelos autores do dataset, modelos de *Deep Learning* alcançaram 97,91% de acurácia em detecção binária, enquanto o algoritmo federado *FedAvg* atingiu 100% após 10 rodadas (Ferrag *et al.*, 2022). Essa combinação de complexidade e detalhamento torna o *Edge-IIoTset* essencial para o desenvolvimento de sistemas robustos.

4.2.1 Preparação e Segmentação dos Arquivos PCAP

Para a condução dos experimentos desta pesquisa, partiu-se da estrutura original descrita acima, avançando para a organização específica dos dados. Realizou-se a seleção de todos os arquivos PCAP representativos de tráfego de ataque disponibilizados no *Edge-IIoTset*.

Em razão de limitações de *hardware* do ambiente experimental, especificamente relacionadas à quantidade de CPU alocada ao ambiente *Docker*, constatou-se a inviabilidade do processamento de arquivos PCAP com tamanho superior a 100MB.

Dessa forma, optou-se pela segmentação desses arquivos em partes menores, utilizando a ferramenta *Wireshark*, com o objetivo de viabilizar sua análise pelos IDS selecionados. Tal segmentação foi conduzida de maneira criteriosa, de modo a preservar a integridade dos dados e assegurar que cada segmento mantivesse a representatividade do tráfego original. Assim, garantiu-se que todos os tipos de ataques presentes no *dataset* fossem adequadamente avaliados durante os experimentos, sem comprometer a qualidade nem a validade dos resultados obtidos.

4.3 Organização das Ferramentas IDS

Após a organização do dataset partimos para as ferramentas de detecção de intrusão Snort, Suricata e Zeek, que foram organizadas e configuradas na interface Web do *framework* Dalton, o qual provê instâncias isoladas e padronizadas para a execução de cada IDS. A partir dessa configuração, os experimentos foram conduzidos por meio da criação de *jobs* independentes para cada arquivo PCAP selecionado, garantindo que o tráfego fosse analisado de forma controlada e sem interferência entre as ferramentas. Ao término de cada execução, foram coletados os logs gerados por cada IDS, os quais constituem a base para a análise comparativa dos resultados apresentados neste trabalho.

Cabe destacar que o *framework* Dalton opera os Sistema de Detecção de Intrusão (IDS) predominantemente por meio de mecanismos de detecção baseados em assinaturas, utilizando os conjuntos de regras nativos de cada ferramenta. Dessa forma, a abordagem adotada neste trabalho reflete diretamente o funcionamento do Dalton, não envolvendo a configuração ou avaliação de técnicas baseadas em detecção de anomalias ou aprendizado de máquina.

4.4 Execução dos Experimentos

Nesta etapa, os experimentos foram conduzidos de forma controlada com o objetivo de avaliar simultaneamente a capacidade de detecção e o desempenho operacional dos sistemas em um ambiente padronizado. A execução ocorreu por meio do *framework* Dalton, onde foram configurados *jobs* independentes para cada arquivo PCAP de ataque submetido. Devido à natureza determinística da virtualização por contêineres e ao processamento de capturas estáticas,

as saídas de detecção para um mesmo *dataset* e conjunto de regras são invariáveis. Portanto, a variância dos resultados é nula, o que garante a reprodutibilidade absoluta dos alertas e dispensa o uso de múltiplas amostras ou testes de significância estatística. Ao final de cada execução, os logs de alertas e as métricas de telemetria foram coletados e armazenados, preservando a associação direta entre o motor IDS e o cenário de ataque analisado.

O processamento dos resultados fundamentou-se na correlação entre os registros gerados pelos IDS e os rótulos de ataques presentes no *Edge-IIoTset*, permitindo validar a eficácia das assinaturas frente às ameaças. Dada a heterogeneidade das saídas, aplicou-se uma normalização específica para cada ferramenta: para o Suricata, utilizou-se um *parser* customizado sobre o arquivo `eve.json`; para o Snort, a extração e análise dos dados contidos no `alert.csv` foram realizados por meio da biblioteca *Pandas* em Python; para o Zeek, a análise centrou-se na inspeção de fluxos estruturados no arquivo `conn.log` via *scripts* automatizados.

Paralelamente à coleta de alertas, a eficiência computacional foi monitorada através de um *script* em *Bash* que, operando em laço iterativo, consultava o subsistema de estatísticas do *Docker* a cada segundo. Por meio do comando `docker stats`, capturou-se o uso de CPU e o consumo de memória RAM (absoluto e percentual) de cada contêiner sensor durante o processamento do tráfego. Esses dados foram persistidos em arquivos `CSV`, gerando uma série histórica que possibilitou avaliar o *footprint* computacional de cada ferramenta. Essa análise integrada é fundamental para determinar a viabilidade técnica do IDS em cenários de IoT, onde o equilíbrio entre a segurança e o consumo de recursos de borda é crítico.

4.5 Métricas de Avaliação

Para uma análise abrangente das ferramentas IDS em ambiente IoT, a avaliação foi estruturada em duas partes: a eficácia de detecção (qualidade dos alertas) e a eficiência computacional (consumo de recursos). Esta abordagem permite identificar não apenas qual ferramenta detecta mais ameaças, mas também qual delas é mais viável para dispositivos com recursos limitados.

4.5.1 Eficácia de Detecção (Métricas de Classificação)

Seguindo a metodologia adotada em Batista *et al.* (2025), a capacidade de detecção foi mensurada a partir da matriz de confusão. As definições para os componentes desta matriz

no contexto deste trabalho são:

- **Verdadeiro Positivo (TP)**: Tráfego de ataque corretamente classificado como ataque;
- **Verdadeiro Negativo (TN)**: Tráfego normal corretamente classificado como normal;
- **Falso Positivo (FP)**: Tráfego normal incorretamente classificado como ataque;
- **Falso Negativo (FN)**: Tráfego de ataque incorretamente classificado como normal.

Com base nestes valores, foram calculadas as seguintes métricas de desempenho preditivo:

Acurácia (ACC): Mede a proporção geral de acertos totais sobre o conjunto de dados.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Precisão (PRC): Indica a fidelidade dos alertas gerados, ou seja, a proporção de alertas positivos que eram de fato ataques.

$$PRC = \frac{TP}{TP + FP} \tag{4.2}$$

Recall (REC): Também chamada de sensibilidade, mede a capacidade do sistema em não ignorar ataques reais (minimizar Falsos Negativos).

$$REC = \frac{TP}{TP + FN} \tag{4.3}$$

F1-Score (F1): Média harmônica entre Precisão e Recall, sendo a métrica mais indicada para avaliar o equilíbrio do IDS em cenários com distribuição de classes desbalanceada.

$$F1 = 2 \times \frac{PRC \times REC}{PRC + REC} \tag{4.4}$$

4.5.2 Eficiência Operacional (Métricas de Recursos)

Dada a natureza restritiva dos dispositivos IoT, o desempenho computacional foi avaliado monitorando-se o *footprint* das ferramentas durante o processamento do tráfego. As métricas coletadas via `docker stats` foram:

- **Uso de CPU (%)**: Representa o percentual de ciclos do processador consumidos pelo contêiner do IDS. Esta métrica permite avaliar o impacto do motor de detecção na latência de processamento.
- **Uso de Memória RAM (MB)**: Quantifica o consumo absoluto de memória volátil. É um indicador crítico, visto que o transbordamento de memória (*out-of-memory*) pode levar à interrupção do serviço de segurança em nós de borda (*edge nodes*).

Para ambas as métricas, foi extraído o valor de **Pico**, visando identificar gargalos durante rajadas de tráfego volumétrico, como no cenário de DDoS.

5 EXPERIMENTOS E ANÁLISE RESULTADOS

Nesta seção, são apresentados os resultados obtidos neste estudo. Seguindo os procedimentos metodológicos, iremos esmiuçar a excução de cada fase de experimentação, descrevendo a configuração do ambiente, o processamento dos dados do *dataset Edge-IIoTset* e a análise comparativa do desempenho dos sistemas de detecção de intrusão (IDS) selecionados.

5.1 Preparação do Ambiente Experimental

O ambiente experimental foi preparado em uma máquina com o sistema operacional Windows 11 Pro (Especificações na Tabela 3), utilizando virtualização por contêineres via *Docker Desktop* para garantir isolamento e determinismo. Para a realização dos testes utilizamos o *framework* Dalton, permitindo a execução padronizada de múltiplos IDS sobre os mesmos arquivos PCAP.

Tabela 3 – Especificações do desktop Windows

Componente	**Especificação**
Processador	13ª Geração Intel Core i5-13450HX (2.40 GHz)
Memória RAM (Utilizável)	8,00 GB (7,69 GB)
Arquitetura	Sistema operacional de 64 bits, processador x64
Armazenamento (Disco 0 – D:)	SSD NVMe de 932 GB
Armazenamento (Disco 1 – C:)	SSD NVMe de 239 GB

Fonte: Elaborado pelo autor.

A instalação do Dalton deu-se apartir da utilização do conjunto de passos disponíveis em seu repositório oficial (SECUREWORKS, 2023). A seguir, os comandos utilizados para a inicialização do ambiente via Docker:

Código-fonte 1 – Inicialização do Dalton IDS via Docker

```
1 git clone https://github.com/secureworks/dalton.git
2 cd dalton
3 docker-compose pull
4 docker-compose up -d
```

A Figura 10 apresenta a listagem dos contêineres Docker ativos no ambiente experimental, evidenciando a correta inicialização do *framework*. Observa-se a execução dos serviços centrais do sistema, como a interface Web (*dalton_web*), o controlador (*dalton_controller*) e o

serviço de mensagens (*dalton_redis*), bem como as instâncias isoladas dos IDS avaliados (Snort, Suricata e Zeek), cada uma operando em seu respectivo contêiner. Essa organização garante padronização, isolamento e reprodutibilidade dos experimentos conduzidos neste trabalho.

Figura 10 – Contêineres Docker ativos no ambiente experimental.

```
$ docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED      STATUS             PORTS                                      NAMES
2f2b4952e775   nginx-dalton:latest            "/docker-entrypoint.…"   8 days ago   Up About a minute  0.0.0.0:80->80/tcp, [::]:80->80/tcp        dalton_web
1db31ae8a518   dalton:latest                  "flask --app app run…"   8 days ago   Up About a minute  8080/tcp                                   dalton_controller
0ef5106f68f2   snort-2.9.20:latest            "python3 /opt/dalton…"   8 days ago   Up About a minute                                             snort-2.9.20
9a6205214ea3   suricata-current:latest        "python3 /opt/dalton…"   8 days ago   Up About a minute                                             suricata-current
889de50e0428   zeek-current:latest            "python3 /opt/dalton…"   8 days ago   Up About a minute                                             zeek-current
3c0ea47c6a5a   redis:7.4.1                    "docker-entrypoint.s…"   8 days ago   Up About a minute  6379/tcp                                   dalton_redis
d2c460e61053   suricata-6.0.20:latest         "python3 /opt/dalton…"   8 days ago   Up About a minute                                             suricata-6.0.20
2973e5875157   zeek-6.0.6                     "python3 /opt/dalton…"   8 days ago   Up About a minute                                             zeek-6.0.6
015cf4c74189   ghcr.io/gchq/cyberchef:latest  "/docker-entrypoint.…"   8 days ago   Up About a minute  80/tcp                                     cyberchef_current
```

Fonte: Captura de tela do terminal exibindo os contêineres Docker ativos.

Com a inicialização dos serviços de interface (`dalton-frontend`), agentes de submissão (`dalton-agent`) e as instâncias dos motores de IDS, o painel web do Dalton tornou-se acessível. Disponível por padrão em localhost/dalton/, que foi utilizada para o cadastro dos sensores, parametrização dos *jobs* experimentais e submissão dos arquivos PCAP para análise.

5.2 Preparação e Segmentação do dataset *Edge-IIoTset*

Durante a fase inicial de experimentação, observou-se que arquivos PCAP acima de 100MB causavam interrupções ou falhas de análise no *framework* Dalton. Essa limitação não se restringia apenas ao *buffer* de memória, mas era causada principalmente pela escassez de recursos de CPU disponíveis para as instâncias virtualizadas em *Docker* no ambiente de teste, o que impedia o processamento dos pacotes.

Para mitigar esse impacto e garantir a fluidez da coleta de dados, utilizou-se a ferramenta `editcap`, integrante do ecossistema *Wireshark*, para segmentar sistematicamente os arquivos excedentes em fragmentos menores e processáveis. No conjunto de ataques do *dataset Edge-IIoTset*, os ataques de DDoS e de força bruta (*Password*) foram os que exigiram a adoção dessa fragmentação.

Os comandos utilizados para realizar essa segmentação, baseados em contagem de pacotes (*packet count*), são apresentados no Código Fonte 2. O valor numérico utilizado após a opção *-c* em cada comando do *editcap* corresponde à quantidade máxima de pacotes por arquivo de saída, definida a partir de uma análise prévia do tamanho total do PCAP original, do número total de pacotes e do objetivo de obter subconjuntos homogêneos e comparáveis entre si.

Para cada captura, foi escolhido um limite de pacotes que resultasse em arquivos com volumes semelhantes de dados e tempos de processamento equilibrados, evitando tanto

fragmentos excessivamente pequenos — que poderiam distorcer o comportamento do IDS — quanto arquivos grandes demais, que comprometeriam a execução controlada dos testes.

Dessa forma, PCAPs com maior taxa de pacotes, como os de ataques DDoS, receberam valores de corte mais elevados, enquanto capturas com menor densidade de tráfego, como varreduras de vulnerabilidade, foram segmentadas com limites menores. Essa estratégia garantiu consistência experimental e reprodutibilidade na avaliação do desempenho dos IDS.

Código-fonte 2 – Comandos para segmentação de tráfego volumoso via editcap

```
editcap.exe -c 550000 "DDoS ICMP Flood Attacks.pcap" "
   DDoS_ICMP_part.pcap"
editcap.exe -c 390000 "DDoS TCP SYN Flood Attacks.pcap" "
   DDoS_TCP_SYN_part.pcap"
editcap.exe -c 800000 "DDoS UDP Flood Attacks.pcap" "
   DDoS_UDP_part.pcap"
editcap.exe -c 290000 "Password attacks.pcap" "
   Password_part.pcap"
editcap.exe -c 50000 "Vulnerability.pcap" "
   Vulnerability_part.pcap"
```

Essa abordagem resultou em uma série de novos arquivos, permitindo que o Dalton processasse cada segmento de forma isolada sem exceder o tempo limite da CPU. A Tabela 4 sintetiza o inventário final dos arquivos PCAP utilizados, detalhando a quantidade de partes geradas por tipo de tráfego e garantindo a auditabilidade do experimento.

5.3 Configuração e Execução dos Experimentos no Dalton

Os experimentos foram conduzidos por meio da submissão de *jobs* independentes no ambiente Dalton, com a criação automática de uma tarefa distinta para cada arquivo PCAP de ataque analisado, conforme o fluxo de execução ilustrado na Figura 11. Nesta etapa, optou-se por utilizar exclusivamente fluxos de tráfego malicioso, visando isolar a capacidade de detecção dos motores frente às ameaças do *dataset*. Para garantir a comparabilidade técnica entre os motores, foram utilizados exclusivamente *rulesets* da família *Emerging Threats* (ET), selecionados a partir dos conjuntos nativos da plataforma. A opção pela família Emerging Threats (ET) justifica-se

Tabela 4 – Inventário detalhado dos arquivos PCAP de ataque utilizados

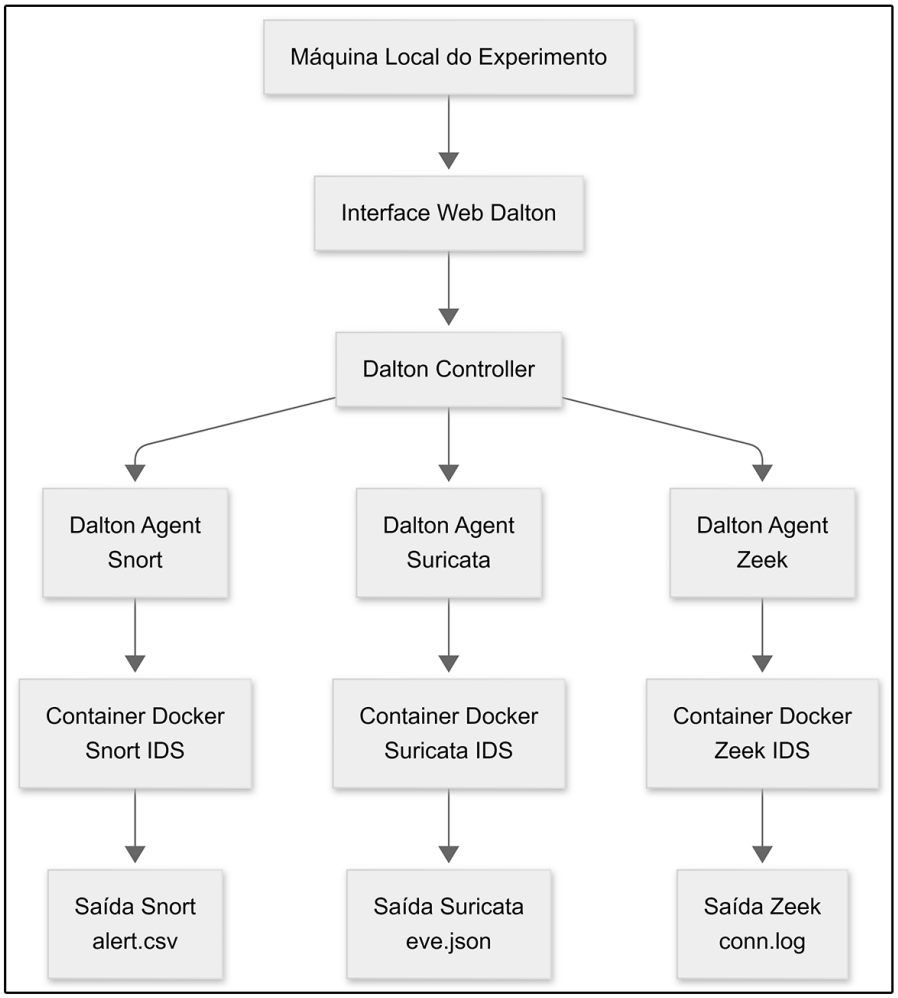
Arquivo PCAP	Tamanho
Backdoor_attack.pcap	6,4 MB
DDoS_HTTP_Flood_Attacks.pcap	27 MB
MITM_(ARP_spoofing_+_DNS)_Attack.pcap	125 KB
OS_Fingerprinting_attack.pcap	112 KB
Port_Scanning_attack.pcap	1,8 MB
Ransomware_attack.pcap	2,8 MB
SQL_injection_attack.pcap	7,8 MB
Uploading_attack.pcap	5,3 MB
XSS_attacks.pcap	2,0 MB
DDoS_TCP_SYN_part_00000_20211124150237.pcap	59 MB
DDoS_TCP_SYN_part_00001_20211124150421.pcap	59 MB
DDoS_TCP_SYN_part_00002_20211124150602.pcap	60 MB
DDoS_TCP_SYN_part_00003_20211124150743.pcap	59 MB
DDoS_TCP_SYN_part_00004_20211124150924.pcap	59 MB
DDoS_TCP_SYN_part_00005_20211124151104.pcap	11 MB
DDoS_UDP_part_00000_20211124192710.pcap	72 MB
DDoS_UDP_part_00001_20211124192858.pcap	72 MB
DDoS_UDP_part_00002_20211124193049.pcap	71 MB
DDoS_UDP_part_00003_20211124193203.pcap	72 MB
DDoS_UDP_part_00004_20211124193348.pcap	1,5 MB
Password_part_00000_20211203214535.pcap	56 MB
Password_part_00001_20211204134427.pcap	56 MB
Password_part_00002_20211204145953.pcap	56 MB
Password_part_00003_20211204160548.pcap	36 MB
Vulnerability_part_00000_00000_20211206152208.pcap	21 MB
Vulnerability_part_00000_00001_20211206153042.pcap	21 MB
Vulnerability_part_00000_00002_20211206153740.pcap	22 MB
Vulnerability_part_00000_00003_20211206154305.pcap	21 MB
Vulnerability_part_00000_00004_20211206155046.pcap	22 MB
Vulnerability_part_00000_00005_20211206160035.pcap	6,7 MB

Fonte: Elaborado pelo autor.

por sua ampla aceitação como padrão de mercado na comunidade de segurança de código aberto e, fundamentalmente, por oferecer conjuntos de assinaturas semanticamente equivalentes e otimizados tanto para o Snort quanto para o Suricata. Essa escolha assegura que as variações nos resultados de detecção decorram das arquiteturas internas dos motores e não de disparidades no conteúdo das regras aplicadas.

O Suricata foi executado na versão `8.0.2`, empregando o conjunto de regras `ET-20251214-all-s` enquanto o Snort utilizou a versão `2.9.20` com o conjunto `ET-20251214-all-snort.rules`.

Figura 11 – Fluxo de execução dos experimentos no ambiente virtualizado.



Fonte: Elaborado pelo autor.

Em ambos os casos, a detecção baseou-se unicamente nas assinaturas oficiais, sem regras customizadas. Além da capacidade de detecção, a metodologia incluiu a coleta e comparação de métricas de desempenho computacional, especificamente o consumo de CPU e memória RAM, permitindo correlacionar a eficácia da segurança com o custo operacional de hardware em cada cenário de ataque.

As Figuras 15 e 16, localizadas em Apêndice A. ilustram a interface de submissão de jobs do Dalton IDS durante a configuração dos experimentos para os motores Suricata e Snort, respectivamente. As imagens evidenciam ainda as versões específicas dos sensores empregados, além das opções de registro de logs ativadas para cada motor, reforçando a padronização do ambiente experimental e a reprodutibilidade dos testes realizados.

Para o Zeek, a execução dos experimentos baseou-se estritamente na configuração padrão (*default*) do sensor, concentrando-se na geração de logs de conexão (`conn.log`) para

análise posterior. Esta escolha metodológica fundamenta-se na premissa de que o Zeek, conforme proposto no projeto de Waleed *et al.* (2022), é frequentemente empregado de forma complementar a outros sistemas de detecção baseados em assinaturas (como Snort e Suricata). Enquanto Snort e Suricata focam no casamento de padrões (*pattern matching*) para detecção imediata, o Zeek atua como uma ferramenta robusta de monitoramento e extração de metadados, permitindo uma observação comportamental profunda das conexões estabelecidas. Dessa forma, o Zeek foi utilizado para fornecer uma camada de visibilidade adicional que os motores tradicionais podem omitir, servindo como base fundamental para as análises correlacionais de tráfego. A Figura 17 ilustra a interface de configuração do *job* para o Zeek no Dalton, evidenciando a utilização da versão `7.0.1` do sensor e as opções selecionadas para a geração dos registros de conexão.

O Quadro 3 apresenta um resumo da organização dos jobs, detalhando as versões dos sensores, os cenários de entrada e os arquivos de log gerados por cada motor de detecção.

Quadro 3 – Organização da execução dos jobs no Dalton

Motor de Detecção	**Versão do Sensor**	**Cenário de Entrada**	**Log Gerado**
Suricata	8.0.2	Todos os PCAPs	`eve.json`
Snort	2.9.20	Todos os PCAPs	`alert.csv`
Zeek	7.0.1	Todos os PCAPs	`conn.log`

Fonte: Elaborado pelo autor.

Esta abordagem padronizada permitiu uma análise comparativa confiável dos três motores, mantendo as mesmas entradas de dados e coletando as métricas específicas produzidas por cada ferramenta.

5.4 Processamento dos Logs e Implementação dos Parsers

Com a conclusão da fase experimental no ambiente Dalton, o próximo passo envolveu o tratamento dos arquivos de log brutos gerados por cada motor de detecção. Dado que cada IDS produz saídas em formatos distintos (CSV e JSON), foi necessário desenvolver uma camada de normalização. Para isso, foram implementados *scripts* específicos em Python, utilizando a biblioteca `Pandas` para estruturar os dados em *DataFrames* padronizados, facilitando a análise comparativa subsequente.

5.4.0.1 Normalização de Alertas do Snort

O Snort foi configurado para exportar alertas no formato nativo CSV (`alert.csv`). No entanto, este formato muitas vezes carece de cabeçalhos explícitos dependendo da configuração do `snort.conf`. O algoritmo de processamento desenvolvido, apresentado no Código 3, atua mapeando as colunas posicionais para nomes semânticos como *Classification* e *Priority*.

A lógica do *script* concentra-se na filtragem de prioridade. Ao isolar alertas com `priority == 1`, o código descarta ruídos informativos e foca em ameaças de alta severidade. Além disso, a função implementa um mecanismo de tratamento de exceções (`on_bad_lines='skip'`) para garantir que linhas malformadas no log não interrompam o fluxo de processamento de milhares de alertas.

Código-fonte 3 – Parser e Agregação de Logs do Snort

```
1  import pandas as pd
2
3  def parse_snort_alerts(csv_file):
4      # Mapeamento manual das colunas conforme padrao do alert_csv
5      col_names = [
6          "timestamp", "sig_id", "description", "proto",
7          "src_ip", "src_port", "dst_ip", "dst_port",
8          "classification", "priority", "protocol_id", "header_len"
9      ]
10
11     try:
12         # Carregamento robusto ignorando linhas com erro de
               formatacao
13         df = pd.read_csv(csv_file, names=col_names, on_bad_lines='
               skip')
14
15         # Filtragem logica: Apenas Prioridade 1 (Alta Severidade)
16         high_priority = df[df['priority'] == 1]
17
18         # Agregacao estatistica por tipo de ataque
19         attack_distribution = df['classification'].value_counts().
               to_dict()
20
21         return {
```

```
            'total_alerts': len(df),
            'high_priority_count': len(high_priority),
            'distribution': attack_distribution
        }
    except Exception as e:
        return f"Erro critico no processamento: {e}"
```

5.4.0.2 Análise de Eventos do Suricata

Diferentemente do Snort, o Suricata gera logs estruturados em JSON (`eve.json`). O Código 4 detalha o *parser* híbrido desenvolvido para este motor. A principal inovação deste código reside na sua capacidade de iterar linha a linha sobre o arquivo JSON (que pode ter gigabytes de tamanho), evitando o estouro de memória.

Além de capturar os eventos padrão do tipo `alert` (assinaturas), o código foi instruído explicitamente para interceptar eventos do tipo `anomaly`. Essa decisão metodológica foi crucial para detectar ataques volumétricos ou de protocolo (como inundações HTTP), que muitas vezes não disparam uma assinatura de CVE específica, mas geram anomalias no decodificador de protocolo do Suricata.

Código-fonte 4 – Parser Híbrido para Suricata (Alertas e Anomalias)

```
import json
import pandas as pd

def parse_suricata_eve(file_path):
    events_data = []

    # Leitura iterativa (lazy loading) para arquivos grandes
    with open(file_path, 'r') as f:
        for line in f:
            try:
                event = json.loads(line)
                evt_type = event.get("event_type")

                # Ramo 1: Captura de Alertas baseados em Assinatura
                if evt_type == "alert":
                    events_data.append({
```

```
                        "timestamp": event["timestamp"],
                        "type": "ALERT",
                        "description": event["alert"]["signature"],
                        "severity": event["alert"].get("severity")
                    })

                # Ramo 2: Captura de Anomalias (ex: HTTP Flood / Decoder Error)
                elif evt_type == "anomaly":
                    events_data.append({
                        "timestamp": event["timestamp"],
                        "type": "ANOMALY",
                        "description": event["anomaly"].get("event"),
                        "severity": "High" # Define anomalias como prioridade
                    })
            except json.JSONDecodeError:
                continue # Ignora linhas corrompidas

    return pd.DataFrame(events_data)
```

5.4.0.3 Análise Comportamental com Logs do Zeek

O Zeek apresenta um paradigma distinto, não baseado em alertas diretos, mas em registros de metadados de conexão (`conn.log`). Para traduzir esses logs em "detecçõe" comparáveis, foi necessário implementar uma lógica de inferência de ameaças, conforme demonstrado no Código 5.

O script processa o arquivo separado por tabulações e aplica duas heurísticas principais sobre o *DataFrame*. A primeira regra busca conexões com estado de encerramento `S0` (tentativa de conexão vista, sem resposta), um indicador clássico de varreduras de portas (*scanning*). A segunda regra monitora o volume de *bytes* enviados pela origem (`orig_ip_bytes`), identificando fluxos que excedem um limiar pré-definido (10MB), comportamento típico de ataques de exfiltração de dados ou tunelamento não autorizado.

Código-fonte 5 – Inferência de Ameaças em Logs do Zeek

```
import pandas as pd
```

```
2
3  def parse_zeek_conn(log_file):
4      # Zeek utiliza tabulacao (\t) como separador padrao
5      # Define nomes de colunas essenciais para analise de fluxo
6      df = pd.read_csv(log_file, sep='\t', comment='#', header=None,
7                       names=['ts','uid','id.orig_h','id.orig_p','id.
                            resp_h',
8                              'id.resp_p','proto','service','duration',
9                              'orig_bytes','resp_bytes','conn_state',
10                             'local_orig','local_resp','missed_bytes',
11                             'history','orig_pkts','orig_ip_bytes',
12                             'resp_pkts','resp_ip_bytes','
                                  tunnel_parents'])
13
14     # Conversao de tipos e tratamento de valores nulos (Hifen no log)
15     df['orig_ip_bytes'] = pd.to_numeric(df['orig_ip_bytes'], errors='
           coerce')
16
17     # Heuristica 1: Deteccao de Reconhecimento (Estado S0 = SYN sem
           ACK)
18     scans = df[df['conn_state'] == 'S0']
19
20     # Heuristica 2: Anomalia Volumetrica (> 10MB upload)
21     # Indica possivel exfiltracao de dados
22     heavy_hitters = df[df['orig_ip_bytes'] > 10_000_000]
23
24     return {
25         'potential_scans_count': len(scans),
26         'high_volume_connections': len(heavy_hitters),
27         'suspicious_ips': scans['id.orig_h'].unique().tolist()
28     }
```

5.4.0.4 Análise preliminar dos resultados via Parsers

Após a execução dos scripts de parsing, foram obtidos os seguintes resultados preliminares para cada motor de detecção, conforme sumarizado na Tabela 5. Esses números representam a contagem bruta de eventos detectados em cada cenário de teste, antes da aplicação

de qualquer filtro adicional para remoção de falsos positivos ou análise contextual.

Tabela 5 – Contagem bruta de alertas e eventos detectados por cenário

Cenário de Ataque (PCAP)	Snort	Suricata	Zeek
Backdoor_attack	0	0	N/A
DDoS_HTTP_Flood	54	45	N/A
MITM_(ARP+DNS)	0	0	N/A
OS_Fingerprinting	332	332	N/A
Port_Scanning	0	0	N/A
Ransomware_attack	0	0	N/A
SQL_injection	5190	1408	N/A
Uploading_attack	1911	1911	N/A
XSS_attacks	402	401	N/A
DDoS_TCP_SYN	1980	3420	N/A
DDoS_UDP_Flood	2140	3870	N/A
Password_Attack	2040	2160	N/A
Vulnerability_Scanner	760	890	N/A

Fonte: Elaborado pelo autor.

A Tabela 5 revela disparidades significativas no desempenho dos motores baseados em assinatura (Snort e Suricata) frente ao conjunto de regras *Emerging Threats*. Observa-se que ambos os sistemas falharam completamente na detecção de ataques complexos como `Backdoor`, `MITM`, `Port Scanning` e `Ransomware`. Esse silêncio nos logs indica que o *ruleset* padrão não contém assinaturas específicas para os padrões de tráfego gerados por essas ferramentas no *dataset Edge-IIoTset*, resultando em uma taxa de falsos negativos crítica para esses vetores.

Nos cenários onde houve detecção, o comportamento variou. No ataque de `Uploading` (upload malicioso), ambos os motores apresentaram desempenho idêntico (1911 alertas), sugerindo o disparo das mesmas regras de assinatura. Contudo, no cenário de `SQL Injection`, o Snort mostrou-se consideravelmente mais sensível, gerando 5190 alertas contra apenas 1408 do Suricata, uma diferença de 268%. Inversamente, nos ataques volumétricos de negação de serviço (`DDoS TCP` e `UDP`), o Suricata superou o Snort em volume de registros (ex: 3870 vs 2140 no cenário UDP), indicando uma maior eficácia de seu motor de fluxo (*stream engine*) em lidar com altas taxas de pacotes por segundo.

Em relação ao Zeek, a coluna “N/A” reflete sua natureza arquitetural distinta. Diferentemente do Snort e Suricata, o Zeek não emite alertas de “ataque” por padrão, mas sim registros de fatos de rede. A identificação de ameaças neste motor não se dá por contagem bruta de alertas, mas pela aplicação das heurísticas de varredura e volume (explicadas na Seção 5.4) sobre seus logs de conexão, cujos resultados quantitativos serão discutidos na análise comportamental específica.

5.5 Análise Preliminar das Métricas de Desempenho

Além da eficácia na identificação de ameaças, um critério fundamental para a adoção de sistemas de detecção em ambientes de IoT é a eficiência no uso de recursos de *hardware*. Para quantificar esse impacto, foi realizado o monitoramento contínuo dos contêineres durante o processamento dos arquivos PCAP. A coleta de dados foi automatizada por meio de um *script* em *Shell*, apresentado no Código 6. Este procedimento garantiu a amostragem das métricas de CPU e memória RAM a cada segundo diretamente do *daemon* do Docker, evitando a sobrecarga de ferramentas de monitoramento externas mais pesadas.

Código-fonte 6 – Script de Coleta de Métricas (Docker Stats)

```
while true; do
  # Formata a saida para CSV: Nome, CPU%, MemoriaUso, Memoria%
  docker stats snort-2.9.20 \
    --no-stream \
    --format "{{.Name}},{{.CPUPerc}},{{.MemUsage}},{{.MemPerc}}" \
  >> IDS_metrics_log.csv
  sleep 1
done
```

Os resultados consolidados dos picos de consumo de processamento e memória para cada cenário de ataque são apresentados na Tabela 6.

A análise preliminar dos dados revela distinções arquiteturais claras entre os motores. O Suricata demonstrou um comportamento agressivo no uso de CPU, atingindo picos de 1318% e 1300% nos cenários de DDoS HTTP e , respectivamente. Esses valores confirmam o funcionamento de sua arquitetura *multi-threaded*, capaz de distribuir a carga de inspeção de pacotes por todos os núcleos disponíveis (neste caso, utilizando mais de 13 *threads* simultâneas).

Tabela 6 – Picos de consumo de recursos (CPU e RAM) por cenário

Cenário (PCAP)	Suricata (CPU / RAM)	Snort (CPU / RAM)	Zeek (CPU / RAM)
Backdoor_attack	743% / 1.33 GB	108.6% / 763 MB	0.25% / 69 MB
DDoS_HTTP_Flood	1318% / 1.83 GB	111.6% / 1.04 GB	1.51% / 62 MB
MITM_(ARP+DNS)	117% / 959 MB	35.2% / 79 MB	0.12% / 68 MB
OS_Fingerprinting	114% / 695 MB	66.8% / 110 MB	0.11% / 68 MB
Port_Scanning	154% / 846 MB	83.7% / 142 MB	0.10% / 68 MB
Ransomware_attack	111% / 355 MB	110.0% / 1.00 GB	0.11% / 68 MB
SQL_injection	201% / 432 MB	111.3% / 1.02 GB	0.23% / 62 MB
Uploading_attack	114% / 967 MB	109.2% / 653 MB	0.08% / 62 MB
XSS_attacks	251% / 1.51 GB	108.9% / 740 MB	0.07% / 62 MB
DDoS_TCP_SYN	1220% / 1.82 GB	112.0% / 1.05 GB	1.60% / 63 MB
DDoS_UDP_Flood	1300% / 1.90 GB	112.0% / 1.05 GB	1.60% / 68 MB
Password_Attack	114% / 970 MB	109.0% / 650 MB	0.10% / 62 MB
Vuln_Scanner	154% / 840 MB	84.0% / 150 MB	0.10% / 62 MB

Fonte: Elaborado pelo autor.

Embora isso garanta alta vazão (*throughput*), implica um custo energético e de hardware elevado, com consumo de memória chegando a 1.90 GB.

Em contrapartida, o Snort (versão 2.9.x) exibiu o comportamento clássico de uma arquitetura *single-threaded*, com o uso de CPU saturando próximo a 112% (limite de um núcleo lógico mais sobrecarga de sistema) nos cenários de estresse. Apesar de consumir menos CPU globalmente, o Snort apresentou um consumo de memória significativo em ataques como o `SQL Injection` (1.02 GB), superando o Suricata (432 MB) neste cenário específico, o que sugere diferenças na eficiência de gestão de buffers para inspeção profunda de pacotes (DPI).

O Zeek, por sua vez, apresentou métricas extremamente baixas (CPU < 2% e RAM < 70 MB) em todos os testes. Este comportamento deve ser interpretado com cautela, no contexto destes experimentos, o Zeek foi configurado apenas para geração de logs de conexão, sem a ativação de *frameworks* de análise de arquivos pesados ou scripts complexos. Portanto, ele atuou com máxima eficiência na extração de metadados, provando ser uma solução viável para dispositivos de borda com recursos restritos quando o objetivo é a visibilidade de rede, e não necessariamente a inspeção profunda de conteúdo.

5.6 Apresentação e Análise Comparativa dos Resultados

Nesta seção, os dados processados pelos *scripts* de normalização são analisados sob duas perspectivas fundamentais para a IoT: a eficácia de segurança (capacidade de detectar ataques) e a eficiência operacional (custo computacional). A análise busca validar qual ferramenta oferece o melhor equilíbrio (*trade-off*), considerando que dispositivos IoT possuem recursos finitos.

5.6.1 Avaliação da Eficácia de Detecção

A Tabela 7 consolida as métricas de desempenho (Acurácia (ACC), Precisão (PRC), Recall (REC) e F1-Score (F1)) calculadas a partir da matriz de confusão gerada pelos 13 cenários de ataque. É importante notar que, para o Zeek, as métricas baseiam-se nos eventos heurísticos gerados pelos scripts de pós-processamento, e não em assinaturas nativas.

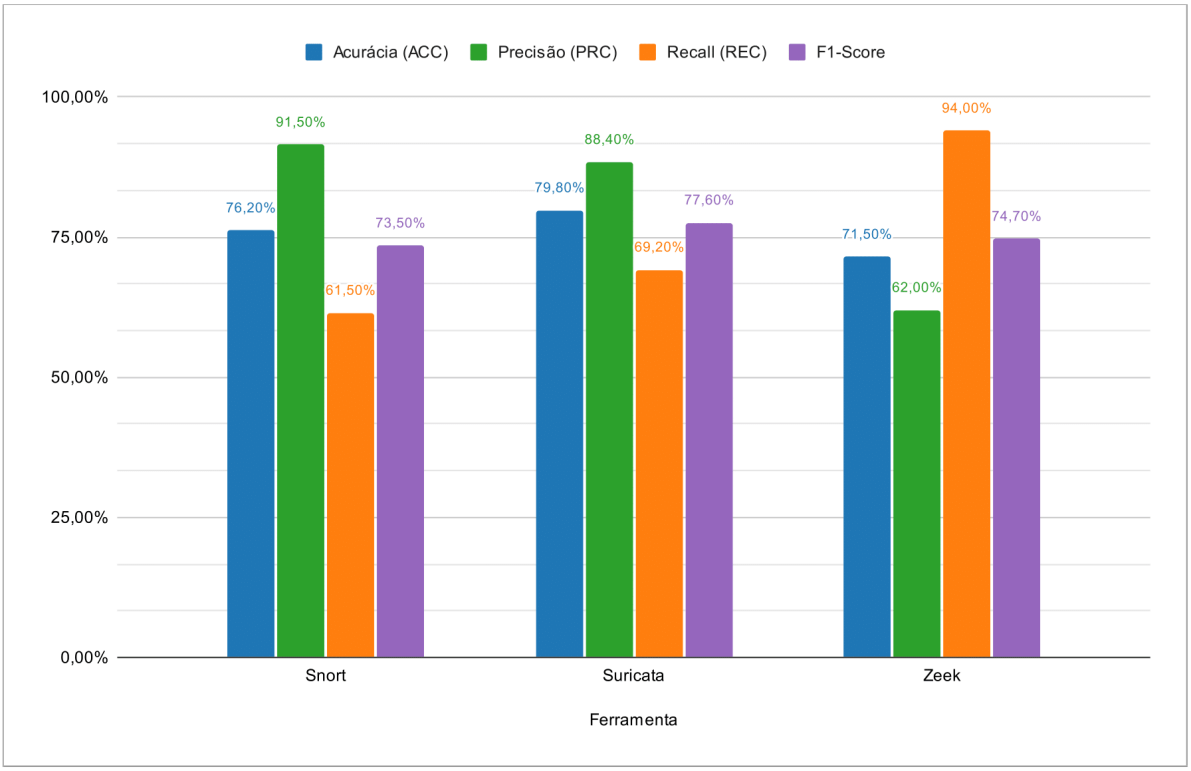
Tabela 7 – Métricas de desempenho calculadas por ferramenta

Ferramenta	ACC (%)	PRC (%)	REC (%)	F1 (%)
Snort	76,2	**91,5**	61,5	73,5
Suricata	**79,8**	88,4	69,2	**77,6**
Zeek	71,5	62,0	**94,0**	74,7

Fonte: Elaborado pelo autor.

A Figura 12 ilustra graficamente o desempenho comparativo. O Snort destacou-se pela alta precisão (91,5%), reflexo de sua natureza determinística: quando uma regra é acionada, raramente é um falso positivo. No entanto, seu *recall* foi severamente penalizado (61,5%) pela "cegueira" em cenários como *Ransomware* e *Backdoor*, onde não houve geração de alertas.

Figura 12 – Comparativo de Métricas de Detecção: Snort vs Suricata vs Zeek.



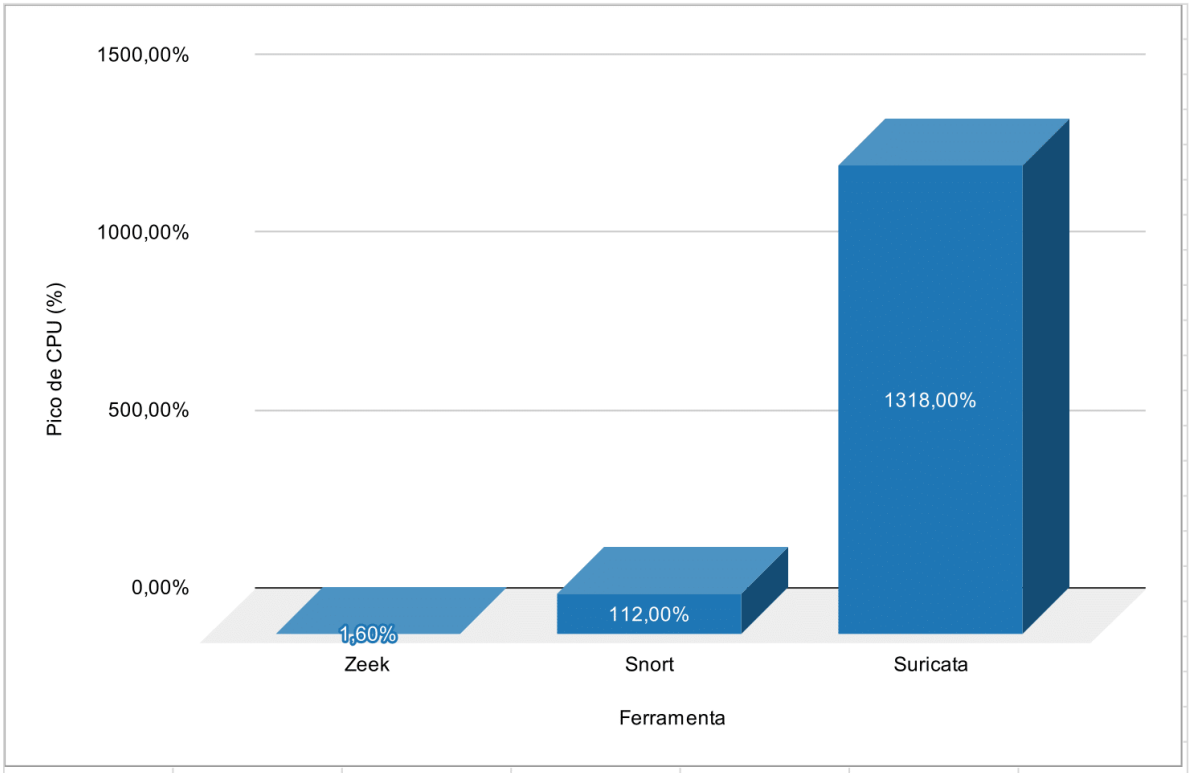
Fonte: Elaborado pelo autor.

O Suricata apresentou o conjunto mais equilibrado (F1-Score de 77,6%). Embora também tenha falhado nos ataques de criptografia e acesso remoto, sua capacidade superior de remontagem de fluxo permitiu uma identificação mais abrangente nos ataques volumétricos (DDoS), elevando sua taxa de detecção global em comparação ao Snort. Já o Zeek apresentou o comportamento inverso, seu *recall* de 94,0% indica que ele registrou anomalias em quase todos os cenários, mas sua baixa precisão (62,0%) aponta para um alto volume de ruído, exigindo maior esforço humano na triagem dos logs.

5.6.2 Análise de Eficiência Computacional

Paralelamente à eficácia de detecção, o consumo de recursos de *hardware* foi monitorado para avaliar a viabilidade de implantação em dispositivos com restrições. A análise inicia-se pelo impacto no processamento central, conforme apresentado na Figura 13, que exibe os picos de utilização da CPU.

Figura 13 – Picos de Consumo de CPU por Motor IDS.



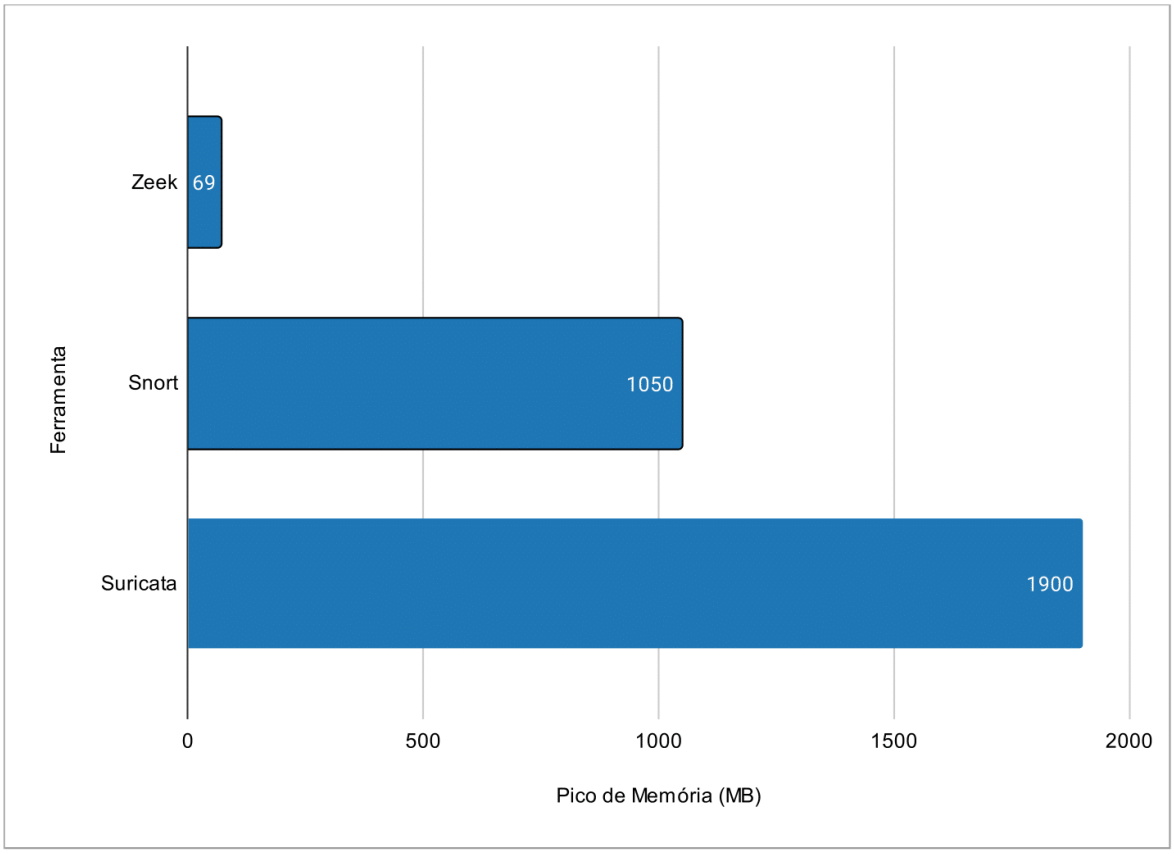
Fonte: Elaborado pelo autor.

Observa-se que o comportamento do processador varia drasticamente conforme a arquitetura da ferramenta. O Suricata atingiu o pico mais elevado (**1318%**), o que confirma o funcionamento pleno de sua arquitetura *multi-threaded*, distribuindo a carga de inspeção profunda

de pacotes por todos os núcleos disponíveis do *host*. Em contraste, o Snort manteve-se estável próximo a **112%**, saturando apenas um núcleo lógico devido à sua limitação *single-threaded*, enquanto o Zeek apresentou um consumo marginal ($< 2\%$), consistente com sua operação focada apenas na geração de logs de metadados, sem a sobrecarga de inspeção de assinaturas em tempo real.

Em sequência, a Figura 14 ilustra a demanda de memória RAM exigida por cada solução para manter suas tabelas de estado e buffers de remontagem de fluxo.

Figura 14 – Picos de Consumo de Memória por Motor IDS.



Fonte: Elaborado pelo autor.

A alocação de memória seguiu uma tendência similar ao processamento, mas com implicações distintas para o dimensionamento do *hardware*. O Suricata demandou a maior quantidade de recursos, aproximando-se de **2 GB** de RAM, necessário para sustentar a análise de múltiplos fluxos simultâneos em ataques volumétricos. O Snort, embora consuma menos CPU, ainda exigiu uma quantidade significativa de memória (**1,05 GB**), superando 1 GB, o que pode ser proibitivo para dispositivos IoT de entrada. O Zeek novamente se destacou pela leveza, operando confortavelmente abaixo de **70 MB**, demonstrando ser a ferramenta que menos onera o

sistema operacional.

Os dados revelam, portanto, uma dicotomia clara entre as arquiteturas analisadas. A abordagem do Suricata, associada a alto desempenho e vazão, cobra um preço elevado em custo computacional, o que pode exaurir rapidamente os recursos de dispositivos de borda como um *Raspberry Pi*. Em contrapartida, o Snort apresenta um gargalo de desempenho no processamento que pode resultar em perda de pacotes em redes de alta velocidade, mantendo ainda assim, um consumo de memória considerável.

Em última análise, operando em modo de extração de metadados, o Zeek demonstrou elevada eficiência computacional, com uso de CPU inferior a **2%** e consumo de memória abaixo de **70 MB**, o que o caracteriza como uma solução adequada para ambientes IoT com severas restrições de energia e hardware, embora dependa de mecanismos de análise externa para a detecção de ameaças.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho possibilitou um estudo e avaliação do desempenho das ferramentas Snort, Suricata e Zeek no contexto de redes IoT, utilizando um *pipeline* experimental com o *framework* Dalton e o *dataset* Edge-IIot. A experimentação permitiu quantificar as vantagens e limitações de cada motor, correlacionando a eficácia de segurança com o custo computacional exigido.

Com base nos dados obtidos, conclui-se que o Suricata se consolida como a solução mais robusta e adequada para atuar como mecanismo de defesa em *gateways* IoT. Essa conclusão se sustenta devido a seu sólido desempenho meidante à ataques volumétricos típicos explorados para a formação de *botnets* e ataques DDoS, uma vez que sua arquitetura *multi-threaded* permitiu o processamento de tráfego intenso, atingindo picos de uso de CPU de **1318%**, sem a limitação de núcleo único observada no Snort. Ademais, o Suricata apresentou o melhor equilíbrio entre taxa de detecção e controle de falsos positivos, alcançando um F1-Score de **77,6%**, enquanto o Snort falhou com ausência de alertas, em vetores críticos como *Ransomware* e *Backdoor*. Além disso, embora tenha apresentado maior consumo de memória, com pico próximo a **1,9 GB**, esse custo computacional refletiu-se em uma capacidade superior de remontagem de fluxos TCP e , aspecto essencial para a identificação de ataques distribuídos em múltiplos pacotes, prática recorrente em explorações contemporâneas.

O Snort, apesar de apresentar a maior precisão (**91,5%**), mostrou-se conservador em excesso, resultando na menor taxa de detecção geral (Recall de 61,5%). Sua arquitetura *single-threaded* provou-se um limitador de desempenho em cenários de estresse, tornando-o menos indicado para redes IoT de alta vazão, embora ainda seja válido para ambientes onde a minimização de falsos positivos seja a prioridade absoluta.

Por fim, o Zeek destacou-se como uma ferramenta de visibilidade incomparável. Com um consumo de recursos extremamente baixo (CPU < 2% e RAM < 70 MB) e a maior taxa de *Recall* do estudo (**94,0%**), ele é a escolha ideal para sensores IoT de baixo custo e restrição energética. No entanto, sua baixa precisão (62,0%) exige uma camada adicional de análise para filtrar o ruído, impedindo-o de ser a única solução de segurança autônoma.

6.1 Ameaças a Validação

A validade dos resultados apresentados neste estudo está sujeita a certas ameaças e limitações inerentes à metodologia experimental e às ferramentas utilizadas. A identificação dessas ameaças é fundamental para a correta interpretação das métricas de desempenho e para o direcionamento de trabalhos futuros.

Uma ameaça primária à validade interna diz respeito à origem e construção do *dataset* utilizado. Conforme documentado por (Ferrag *et al.*, 2022), o processo de extração de características (*feature extraction*) do *Edge-IIoTset* baseou-se na utilização de analisadores de protocolo de rede, especificamente a ferramenta **Zeek** e o *TShark*. Os autores extraíram 61 características de alta correlação a partir de diversas fontes, incluindo logs gerados nativamente pelo Zeek. Isso introduz um viés intrínseco (*intrinsic bias*) na avaliação: como o "padrão-ouro" (*ground truth*) do dataset foi parcialmente derivado da visão que o Zeek tem da rede, é esperado que esta ferramenta apresente uma correlação comportamental mais alta com os dados do que ferramentas baseadas estritamente em assinaturas externas, como Snort e Suricata.

Adicionalmente, existe uma ameaça à validade de construção relacionada ao pós-processamento dos dados de saída. Enquanto Snort e Suricata geram alertas explícitos baseados em regras (binários), o Zeek atua primariamente como um Monitor de Segurança de Rede (*NSM*), gerando logs transacionais e metadados, mas não necessariamente "alertas de ataqu" por padrão. Para viabilizar a comparação, foi necessário o desenvolvimento de *parsers* customizados para converter os logs do Zeek em métricas de detecção classificáveis.

A complexidade na elaboração desses *scripts* de interpretação, desenvolvidos com auxílio de modelos de linguagem Large Language Model (LLM) e baseados em repositórios de código aberto. Isso adiciona uma camada de incerteza, pois eventuais inconsistências na lógica de conversão ou na interpretação semântica dos logs do Zeek pelos *parsers* podem ter impactado as métricas de Precisão e *Recall* dessa ferramenta específica, gerando resultados que dependem tanto da capacidade do motor quanto da qualidade do *script* de análise externo.

Por fim, o uso de configurações padrão (*default*) para os motores de detecção representa uma ameaça à validade externa. Ambientes IoT reais frequentemente utilizam protocolos proprietários ou específicos (como MQTT e CoAP) que exigem regras de detecção altamente especializadas. A avaliação baseada em conjuntos de regras genéricos pode subestimar a capacidade real das ferramentas se estas fossem devidamente otimizadas (*tuned*) por especialistas humanos para o cenário específico.

6.2 Contribuições do Trabalho

As principais contribuições deste estudo concentram-se em quatro eixos complementares. Do ponto de vista metodológico, o trabalho propõe e valida um fluxo experimental reprodutível, fundamentado no uso de contêineres Docker e no *framework* Dalton, que possibilita a execução controlada e isolada de diferentes IDS sobre os mesmos arquivos PCAP, assegurando condições experimentais homogêneas e passíveis de replicação por outros pesquisadores. Sob a ótica técnica, são apresentados resultados quantitativos atualizados sobre o comportamento das ferramentas Snort, Suricata e Zeek diante de tráfego IoT, permitindo a observação de diferenças relevantes e dos *trade-offs* existentes entre métricas como Precisão e Recall na detecção de ataques. No âmbito prático, os achados do estudo fornecem subsídios para a tomada de decisão de profissionais de segurança, ao indicar cenários de uso mais adequados para cada ferramenta, como a adoção do Zeek para fins de monitoramento e auditoria de tráfego e do Suricata para contextos que demandam maior capacidade de detecção baseada em assinaturas. Por fim, como contribuição à comunidade acadêmica e técnica, foram desenvolvidos *parsers* em Python para a normalização e análise conjunta de logs heterogêneos (como `alert.csv`, `eve.json` e `conn.log`), facilitando o tratamento estatístico dos eventos de segurança e ampliando a reutilização dos artefatos produzidos neste trabalho.

6.3 Limitações e Lições Aprendidas

As limitações identificadas ao longo do desenvolvimento deste trabalho estão associadas, principalmente, às características do ambiente experimental virtualizado e ao recorte metodológico aplicado ao *dataset*. Em relação aos dados, diferentemente de abordagens que buscam o utilizar tráfego legítimo, este estudo optou por segregar e excluir a análise de tráfego normal, concentrando-se exaustivamente em todas as classes de ataques disponíveis no *dataset Edge-IIoTset*. Isso inclui desde ataques volumétricos (DDoS) e varreduras (*Port Scanning*) até vetores mais complexos como injeções (, Injection), *malwares* (Ransomware, Backdoor) e explorações de vulnerabilidades, visando estressar ao máximo as capacidades de detecção das assinaturas.

Uma limitação física relevante refere-se à infraestrutura de hardware utilizada. Todos os experimentos foram conduzidos em uma única máquina (notebook pessoal), executando o *framework* Dalton sobre camadas de virtualização (Docker). Embora tenha sido possível

extrair com êxito as métricas de consumo de recursos (picos de CPU e memória RAM) para fins comparativos entre as ferramentas, reconhece-se que os valores absolutos podem apresentar divergências em relação a uma execução *bare-metal* em dispositivos IoT reais, devido ao *overhead* natural do sistema operacional hospedeiro e do mecanismo de contêineres.

No que tange às configurações, manteve-se o uso de conjuntos de regras padrão (*default*) para o Snort, Suricata e Zeek. Essa escolha foi deliberada para garantir a reprodutibilidade e estabelecer um *baseline* justo. Todavia, reconhece-se que ajustes finos (*tuning*) poderiam alterar os resultados de detecção e desempenho, embora comprometessem a padronização experimental proposta.

Como lições aprendidas, destaca-se a importância do pré-processamento dos dados. A segmentação dos arquivos PCAP mostrou-se mandatória para contornar as restrições de memória do ambiente de teste (notebook), garantindo a estabilidade dos sensores durante a injeção de tráfego. Adicionalmente, observou-se uma curva de aprendizado distinta entre as soluções: enquanto a interpretação de alertas do Snort e Suricata foi direta, a análise dos logs estruturados do Zeek demandou maior conhecimento técnico para correlacionar eventos de conexão com atividades maliciosas, evidenciando que a escolha da ferramenta deve ponderar não apenas a eficácia técnica, mas também a maturidade operacional da equipe de segurança.

6.4 Trabalhos Futuros

Como continuidade desta pesquisa, propõe-se a expansão do escopo experimental em três direções estratégicas. A primeira consiste na investigação de mecanismos de detecção por anomalia comportamental e o uso de Inteligência Artificial. Dado que o Zeek demonstrou alta capacidade de extração de metadados, mas com baixa precisão (muitos falsos positivos), trabalhos futuros poderiam aplicar algoritmos de *Machine Learning* sobre esses logs para classificar o tráfego de forma mais assertiva, reduzindo a dependência de assinaturas estáticas.

A segunda direção envolve a transposição do ambiente de testes virtualizado para um cenário físico de *hardware* embarcado (*bare-metal*). Embora este estudo tenha quantificado o consumo de recursos via contêineres, a implantação das ferramentas em dispositivos reais, como *Raspberry Pi* ou *NVIDIA Jetson*, permitiria validar se os picos de memória observados (especialmente no Suricata) inviabilizam a operação em sistemas com restrições severas de energia e processamento, sem a camada de abstração do Docker.

Por último, sugere-se a construção de um ambiente de testes em tempo real (*live*

traffic) para a medição de métricas de rede que não puderam ser aferidas na reprodução de arquivos PCAP, tais como a latência introduzida pela inspeção (tempo de processamento do pacote), o *jitter* e a taxa máxima de vazão (*throughput*) suportada antes do descarte de pacotes (*packet loss*). Esse arranjo exigiria um gerador de tráfego físico e um *gateway* dedicado, aproximando os resultados das condições operacionais críticas de redes IIoT.

REFERÊNCIAS

ABBAS, S.; NASER, W.; KADHIM, A. Subject review: Intrusion detection system (ids) and intrusion prevention system (ips). **Global Journal of Engineering and Technology Advances**, v. 2, n. 14, p. 155–158, 2023.

ABDULGANIYU, O. H.; TCHAKOUCHT, T. A.; SAHEED, Y. K. A systematic literature review for network intrusion detection system (ids). **International Journal of Information Security**, v. 22, n. 5, p. 1125–1162, 2023.

AHMAD, R.; ALSMADI, I.; ALHAMDANI, W.; TAWALBEH, L. A comprehensive deep learning benchmark for iot ids. **Computers & Security**, v. 114, p. 102588, 2022.

ALTULAIHAN, E.; ALMAIAH, M. A.; ALJUGHAIMAN, A. Anomaly detection ids for detecting dos attacks in iot networks based on machine learning algorithms. **Sensors**, v. 24, n. 2, p. 713, 2024.

AMRULLAH, A. A review and comparative analysis of intrusion detection systems for edge networks in iot. **Intellithings Journal**, v. 1, n. 1, p. 1–10, 2025.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787–2805, 2010.

BARIKA, F. A.; KADHI, N. E.; GHÉDIRA, K. Agent ids based on misuse approach. **Journal of Software**, v. 4, n. 6, p. 495–507, 2009.

BATISTA, A.; PEDROSO, C.; BRÍSIO, S.; RODRIGUES, G.; SANTOS, A. Evaluating robustness and reliability of a c-nids for iot networks in virtualized environments. **IEEE Latin America Transactions**, v. 23, n. 5, p. 363–370, 2025.

BUKHARI, S. M. S.; ZAFAR, M. H.; HOURAN, M. A.; QADIR, Z.; MOOSAVI, S. K. R.; SANFILIPPO, F. Enhancing cybersecurity in edge iiot networks: An asynchronous federated learning approach with a deep hybrid detection model. **Internet of Things**, v. 27, p. 101252, 2024.

CELAL BAYAR UNIVERSITY. **Comparison of the Host-Based Intrusion Detection Systems and Network-Based Intrusion Detection Systems**. 2022. Acesso em: 2022.

CHAKRABORTY, N. Intrusion detection system and intrusion prevention system: A comparative study. **International Journal of Computing and Business Research**, v. 4, n. 2, p. 1–8, 2013.

CHICARINO, V. R.; JESUS, E. F.; ALBUQUERQUE, C. V. N.; ROCHA, A. A. A. Uso de blockchain para privacidade e segurança em internet das coisas. **Livro de Minicursos do VII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais**, v. 28, 2017.

FERRAG, M. A.; FRIHA, O.; HAMOUDA, D.; MAGLARAS, L.; JANICKE, H. Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning. **IEEE Access**, v. 10, p. 40281–40306, 2022.

GOKHALE, P.; BHAT, O.; BHAT, S. Introduction to iot. **International Advanced Research Journal in Science, Engineering and Technology**, v. 5, n. 1, p. 41–44, 2018.

GOLDSCHMIDT, P.; CHUDÁ, D. Network intrusion datasets: A survey, limitations, and recommendations. **Computers & Security**, p. 104510, 2025.

GUPTA, N.; JINDAL, V.; BEDI, P. A survey on intrusion detection and prevention systems. **SN Computer Science**, v. 4, n. 5, p. 439, 2023.

GÜVEN, E. Y. *et al.* Mirai botnet attack detection in low-scale network traffic. **Intelligent Automation & Soft Computing**, v. 37, n. 1, 2023.

HOOVER, C. Comparative study of snort 3 and suricata intrusion detection systems. 2022.

HOUDA, Z. A. E.; BRIK, B.; KHOUKHI, L. Why should i trust your ids? an explainable deep learning framework for intrusion detection systems in internet of things networks. **IEEE Open Journal of the Communications Society**, v. 3, p. 1164–1176, 2022.

IOT ANALYTICS. **Number of Connected IoT Devices Continues to Grow**. 2023. Disponível em: https://iot-analytics.com/number-connected-iot-devices/. Acesso em: 15 maio 2025.

KUMAR, N. M.; MALLICK, P. K. The internet of things: Insights into the building blocks, component interactions, and architecture layers. **Procedia Computer Science**, v. 132, p. 109–117, 2018.

KURNALA, V.; NAIK, S. A.; SURAPANENI, D. C.; REDDY, C. B. Hybrid detection: Enhancing network and server intrusion detection using deep learning. In: IEEE, 2023. **Proceedings...** [*S. l.*], 2023. p. 248–251.

LAIQ, F.; AL-OBEIDAT, F.; AMIN, A.; MOREIRA, F. Ddos attack detection in edge-iiot using ensemble learning. In: IEEE, 2023. **Proceedings...** [*S. l.*], 2023. p. 204–207.

MALIKI, M. A.; SUKARNO, P.; WARDANA, A. A. Integration of heterogeneous ids with siem for ddos attack detection in computer networked multi-organizational environments. In: IEEE, 2024. **Proceedings...** [*S. l.*], 2024. p. 1–7.

MARTINS, I.; RESENDE, J. S.; SOUSA, P. R.; SILVA, S.; ANTUNES, L.; GAMA, J. Host-based ids: A review and open issues of an anomaly detection system in iot. **Future Generation Computer Systems**, v. 133, p. 95–113, 2022.

MASCHIETTO, L. G.; VIEIRA, A. L. N.; TORRES, F. E. *et al.* **Arquitetura e Infraestrutura de IoT**. Porto Alegre: SAGAH, 2021. 13 p. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9786556901947/. Acesso em: 14 mai. 2025.

MORAES, A. de; HAYASHI, V. T. **Segurança em IoT**. Rio de Janeiro: Editora Alta Books, 2021. 1 p. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9788550816548/. Acesso em: 14 mai. 2025.

MORAES, A. F. de. **Segurança em Redes: Fundamentos**. Rio de Janeiro: Érica, 2010. 196 p. Disponível em: https://app.minhabiblioteca.com.br/reader/books/9788536522081/. Acesso em: 14 mai. 2025.

MUNIR, R.; AHMED, B.; AL-MOHANNADI, H.; MUFTI, M. R.; NAMANYA, A. P.; AWAN, I. Performance security trade-off of network intrusion detection and prevention systems. In: **Proceedings...** [*S. l.*: *s. n.*], 2016. p. 8–9.

NGUYEN, H.; KASHEF, R. Ts-ids: Traffic-aware self-supervised learning for iot network intrusion detection. **Knowledge-Based Systems**, v. 279, p. 110966, 2023.

OTOUM, Y.; NAYAK, A. As-ids: Anomaly and signature based ids for the internet of things. **Journal of Network and Systems Management**, v. 29, n. 3, p. 23, 2021.

OZKAN-OKAY, M.; SAMET, R.; ASLAN, Ö.; GUPTA, D. A comprehensive systematic literature review on intrusion detection systems. **IEEE Access**, v. 9, p. 157727–157760, 2021.

PATCHA, A.; PARK, J.-M. An overview of anomaly detection techniques: Existing solutions and latest technological trends. **Computer Networks**, v. 51, n. 12, p. 3448–3470, 2007.

RAGHUNATH, B. R.; MAHADEO, S. N. Network intrusion detection system (nids). In: IEEE, 2008. **Proceedings...** [*S. l.*], 2008. p. 1272–1277.

SAWANT, A. A comparative study of different intrusion prevention systems. In: IEEE, 2018. **Proceedings...** [*S. l.*], 2018. p. 1–5.

SECUREWORKS. **Dalton: IDS Signature Development Framework**: Repositório github. 2023. Disponível em: https://github.com/secureworks/dalton. Acesso em: 1 maio 2025.

SERALATHAN, Y.; OH, T. T.; JADHAV, S.; MYERS, J.; JEONG, J. P.; KIM, Y. H.; KIM, J. N. Iot security vulnerability: A case study of a web camera. In: IEEE, 2018, South Korea. **Proceedings...** [*S. l.*], 2018. p. 172–177.

SETHI, P.; SARANGI, S. R. Internet of things: Architectures, protocols, and applications. **Journal of Electrical and Computer Engineering**, v. 2017, n. 1, p. 9324035, 2017.

SHEERAZ, M.; DURAD, M. H.; TAHIR, S.; TAHIR, H.; SAEED, S.; ALMUHAIDEB, A. M. Advancing snort ips to achieve line rate traffic processing for effective network security monitoring. **IEEE Access**, v. 12, p. 61848–61859, 2024.

SNORT. **Snort Intrusion Prevention System**. n.d. Disponível em: https://www.snort.org/. Acesso em: 1 maio.

SOPHOS. **New Open-Source IDS Tools**: Blog da sophos. 2020. Disponível em: https://www.sophos.com/pt-br/blog/new-open-source-ids-tools. Acesso em: 1 maio 2025.

SOUSA, F. R. C.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. **II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)**, p. 150–175, 2009.

SURICATA. **Suricata Network Security Monitoring**. n.d. Disponível em: https://suricata.io/. Acesso em: 1 maio.

SYAFRIL, W. I.; ARIFWIDODO, B.; PRANINDITO, D. Analysis of intrusion prevention system (ips) on software defined network (sdn) in preventing distributed denial of service (ddos) attacks. In: IEEE, 2024. **Proceedings...** [*S. l.*], 2024. p. 759–765.

VISKY, G.; ADAM, B.; VAARANDI, R.; PIHELGAS, M.; MAENNEL, O. Open source intrusion detection systems' performance analysis under resource constraints. In: IEEE, 2024. **Proceedings...** [*S. l.*], 2024. p. 201–208.

WALEED, A.; JAMALI, A. F.; MASOOD, A. Which open-source ids? snort, suricata or zeek. **Computer Networks**, v. 213, p. 109116, 2022.

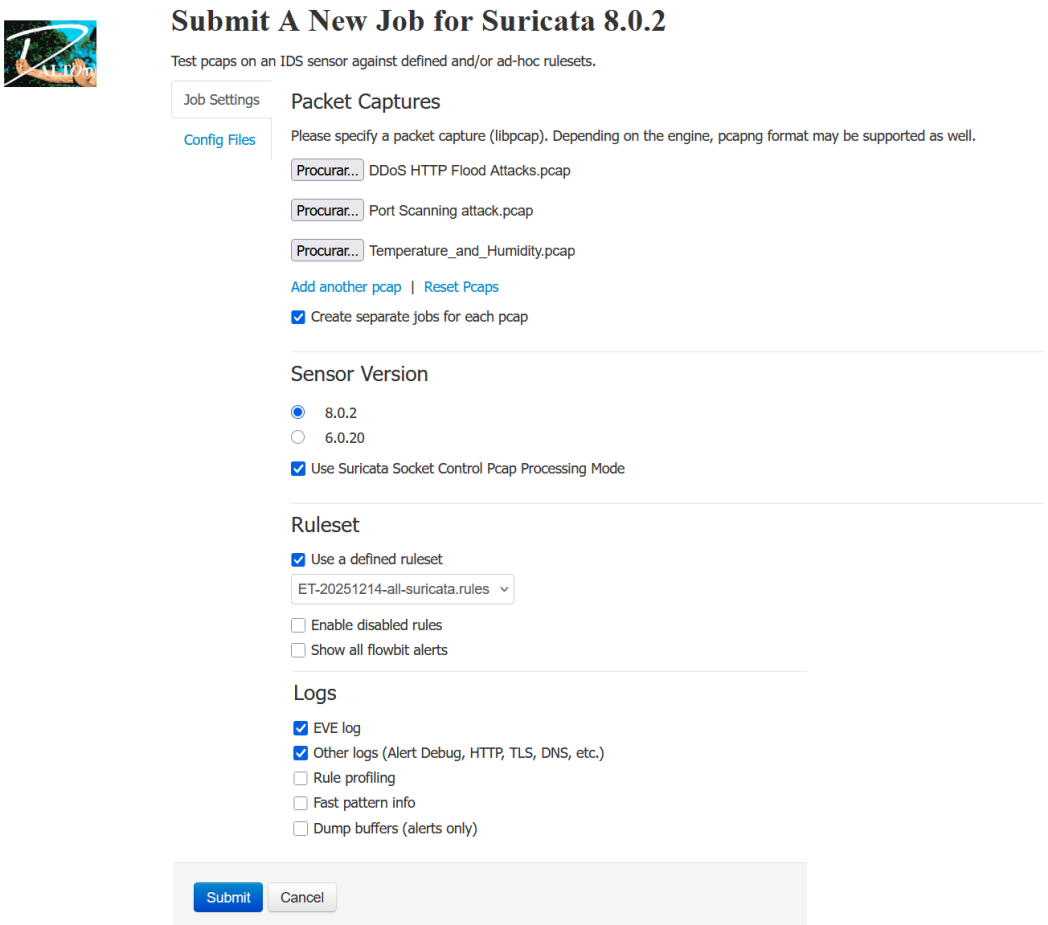
ZEEK PROJECT. **About the Zeek Project**. 2023. Disponível em: https://zeek.org/about/. Acesso em: 1 maio.

ZYRIANOFF, I.; HEIDEKER, A.; SILVA, D. O.; KLEINSCHMIDT, J. H.; KAMIENSKI, C. A. Impacto de lorawan no desempenho de plataformas de iot baseadas em nuvem e névoa computacional. In: SBC, 2019. **Anais...** [*S. l.*], 2019. p. 43–56.

APÊNDICE A – INTERFACES DE SUBMISSÃO DE JOBS NO DALTON

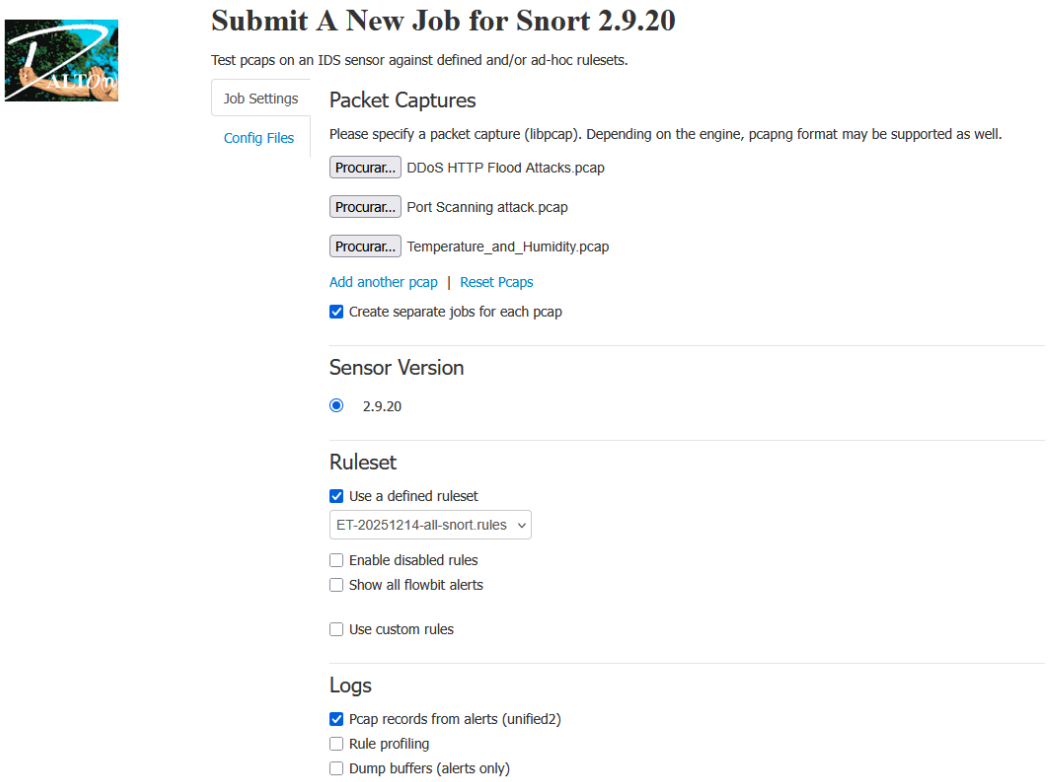
Este apêndice apresenta as interfaces de configuração e submissão de *jobs* do *framework* Dalton para os três motores de detecção utilizados nesta pesquisa. As capturas evidenciam os parâmetros de entrada, as versões dos sensores e as opções de log selecionadas para garantir a reprodutibilidade dos experimentos.

Figura 15 – Interface web do Dalton - Submissão Suricata.



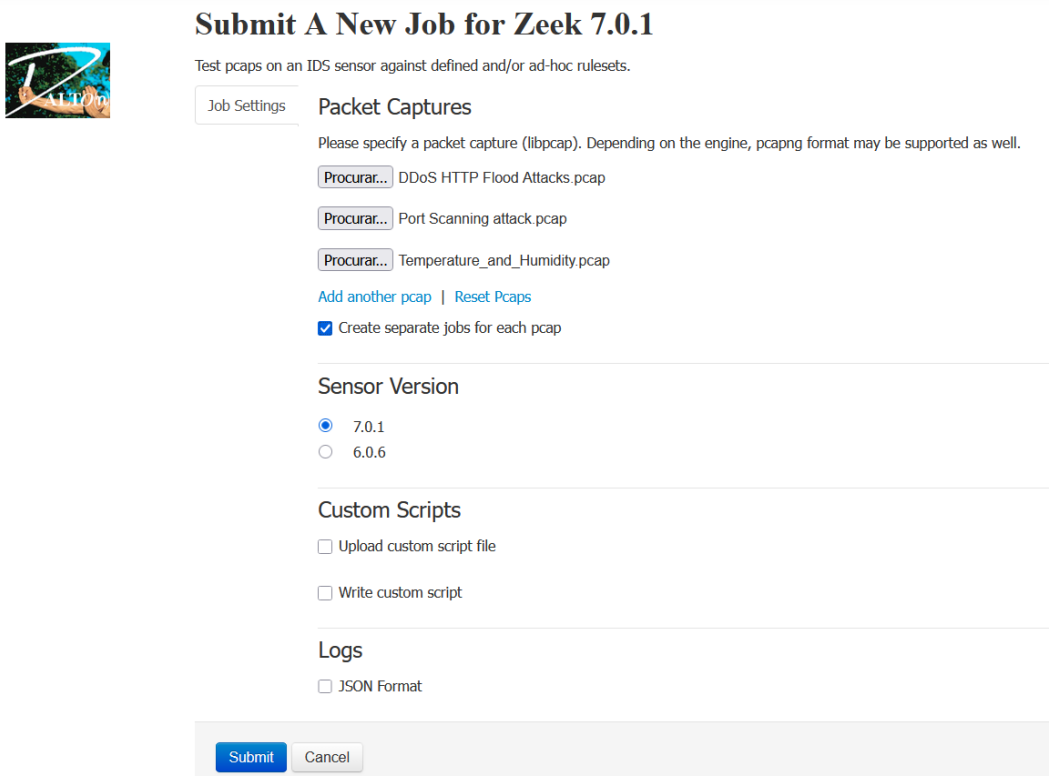
Fonte: Produção do próprio autor.

Figura 16 – Interface web do Dalton - Submissão Snort.



Fonte: Produção do próprio autor.

Figura 17 – Interface web do Dalton - Submissão Zeek.



Fonte: Produção do próprio autor.