

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA

Relatório de Estágio Supervisionado

Jan Paulo de Lima Filho

Campina Grande - PB

Março de 2026

Jan Paulo de Lima Filho

Relatório de Estágio Supervisionado

Relatório de Estágio Supervisionado submetido à Coordenadoria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Grau de Engenheiro Eletricista.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Unidade Acadêmica de Engenharia Elétrica - UAEE

Coordenação de Graduação em Engenharia Elétrica - CGEE

Gutemberg Gonçalves dos Santos Júnior, Dr.

(Orientador)

Campina Grande - PB

Março de 2026

Jan Paulo de Lima Filho

Relatório de Estágio Supervisionado

Relatório de Estágio Supervisionado submetido à Coordenadoria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Grau de Engenheiro Eletricista.

Aprovado em ____ / ____ / ____

Professor Avaliador

Universidade Federal de Campina Grande
Avaliador

**Gutemberg Gonçalves dos Santos
Júnior**

Universidade Federal de Campina Grande
Orientador

Campina Grande - PB
Março de 2026

"Dedico este trabalho à minha família de origem, com meu pai, minha mãe e minha irmã, por todas as vivências que contribuíram para meu amadurecimento. E à família que ainda vou construir, com minha namorada e nossos futuros filhos, que já me motivam mesmo antes de existirem."

Agradecimentos

Agradeço aos amigos e familiares que se mantiveram próximos durante essa jornada.

“I did it my way.”
(Frank Sinatra, "My Way", 1969)

Resumo

A crescente complexidade de sistemas conectados em rede amplia a necessidade de mecanismos capazes de identificar comportamentos que se afastem do funcionamento esperado, inclusive em situações nas quais não há assinaturas ou regras específicas previamente definidas para cada ameaça. Nesse contexto, este trabalho apresenta uma prova de conceito para detecção de anomalias em um ambiente experimental que simula um sistema distribuído de dispositivos, considerando duas perspectivas complementares de observação: nível de rede e nível de aplicação. A proposta baseia-se na modelagem do comportamento benigno do sistema a partir de métricas extraídas dos fluxos de comunicação e dos registros de requisições processadas pela aplicação central, utilizando modelos Autoencoder treinados exclusivamente com dados normais para identificar desvios em janelas temporais. Os experimentos contemplaram cenários como reconhecimento de rede, força bruta em chave de API, enumeração de endpoints e exploração de fraquezas de código, permitindo analisar como diferentes tipos de anomalia se manifestam em cada nível de monitoramento. Os resultados demonstram que a abordagem foi capaz de identificar comportamentos anômalos no sistema e evidenciam que as perspectivas de rede e aplicação se complementam, fornecendo uma visão mais abrangente do comportamento observado em sistemas distribuídos.

Palavras-chave: Detecção de anomalias, Autoencoder, Monitoramento de rede, Monitoramento de aplicação, Sistemas distribuídos, Análise comportamental.

Abstract

The increasing complexity of systems connected to a network highlights the need for mechanisms capable of identifying behaviors that deviate from expected operation, especially in scenarios where no predefined signatures or rules exist for specific threats. In this context, this work presents a proof of concept for anomaly detection in an experimental environment that simulates a distributed device system, considering two complementary monitoring perspectives: network-level and application-level. The proposed approach is based on modeling the benign behavior of the system using metrics extracted from communication flows and request records processed by the central application, employing Autoencoder models trained exclusively on normal data to identify deviations over time windows. The experiments considered scenarios such as network reconnaissance, API key brute force, endpoint enumeration, and exploitation of code weaknesses, allowing the analysis of how different types of anomalies manifest at each monitoring level. The results demonstrate that the approach was able to identify anomalous behaviors in the system and highlight that network and application perspectives are complementary, providing a more comprehensive view of the observed behavior in distributed systems.

Keywords: Anomaly detection, Autoencoder, Network monitoring, Application monitoring, Distributed systems, Behavioral analysis.

Lista de ilustrações

Figura 1 – Envio de um comando ao sistema para acionar uma lâmpada, com resposta do hub após o encaminhamento ao dispositivo correspondente	18
Figura 2 – Esquema geral da arquitetura do Autoencoder para o nível de rede . .	27
Figura 3 – Anomalia detectada em janela de 10s durante a varredura do Nmap . .	32
Figura 4 – Exemplo de janela de 10s com mais de 2500 fluxos observados durante o reconhecimento de rede com Nmap	32
Figura 5 – Exemplos de métricas <i>app-level</i> para janela de 10s durante o ataque de <i>Brute Force</i>	33
Figura 6 – Registros de auditoria do hub durante tentativas de acesso de endpoints inexistentes	34
Figura 7 – Métricas de comando benigno e malicioso no endpoint <code>/demo/search</code> .	36

Lista de abreviaturas e siglas

API	<i>Application Programming Interface</i>
CSV	<i>Comma-Separated Values</i>
CWE	<i>Common Weakness Enumeration</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IoT	<i>Internet of Things</i>
JSON	<i>JavaScript Object Notation</i>
RCE	<i>Remote Code Execution</i>
SSTI	<i>Server-Side Template Injection</i>
TCP	<i>Transmission Control Protocol</i>

Sumário

1	INTRODUÇÃO	12
1.1	Local do Estágio	12
1.2	Objetivos	12
1.3	Organização do Trabalho	13
2	CONCEITOS E DEFINIÇÕES	14
2.1	Segurança de Sistemas e Ameaças	14
2.2	Monitoramento de Tráfego	14
2.3	Deteccção de Anomalias	15
2.4	Autoencoders	15
3	ATIVIDADES REALIZADAS	17
3.1	Criação do Ambiente Experimental	17
3.1.1	Desenvolvimento do Hub e Dispositivos	17
3.1.2	Fluxos e Interações	18
3.2	Definição de Métricas	19
3.2.1	Métricas em Nível de Rede	19
3.2.2	Métricas em Nível de Aplicação	20
3.3	Modelagem do Comportamento Base	21
3.3.1	Definição de Comportamento Base	21
3.3.2	Rotinas para Geração de Tráfego Benigno	23
3.4	Coleta de Dados	23
3.4.1	Coleta em Nível de Rede	23
3.4.2	Coleta em Nível de Aplicação	24
3.5	Construção do Dataset	24
3.5.1	Agrupamento Temporal	24
3.5.2	Construção das Features	25
3.6	Treinamento dos Modelos	26
3.7	Monitoramento e Inferência em Tempo Real	27
3.7.1	Pipeline de Monitoramento	28
3.7.2	Inferência Contínua	28
3.8	Cenários de Anomalia	29
3.8.1	Network Reconnaissance	29
3.8.2	Brute Force de API Key	29
3.8.3	Enumeration Attack	29
3.8.4	SSTI (CWE-1336) / RCE	30

4	RESULTADOS	31
4.1	Caracterização do Comportamento Base	31
4.2	Deteccção de Anomalias	31
4.2.1	Network Reconnaissance	32
4.2.2	Brute Force de API Key	33
4.2.3	Enumeration Attack	34
4.2.4	SSTI (CWE-1336) / RCE	35
4.3	Análise em Níveis: Rede e Aplicação	36
4.4	Discussões	37
4.4.1	Limitações	37
4.4.2	Trabalhos Futuros	38
5	CONCLUSÃO	39
	REFERÊNCIAS	40

1 Introdução

A crescente complexidade dos sistemas conectados em rede amplia a necessidade de mecanismos capazes de observar seu funcionamento e identificar comportamentos que se afastem do padrão esperado. Em muitos casos, atividades maliciosas ou indevidas não se manifestam apenas pela exploração direta de falhas conhecidas, mas também por alterações no modo como serviços e componentes se comunicam ao longo do tempo. Nesse contexto, abordagens baseadas em comportamento tornam-se relevantes por permitirem a identificação de desvios mesmo quando não há padrões explícitos ou regras previamente definidas para cada ameaça.

Este trabalho investiga a detecção de anomalias em um ambiente experimental que simula um sistema distribuído de dispositivos, considerando duas perspectivas complementares de monitoramento: nível de rede e nível de aplicação. A proposta envolve a definição de métricas representativas do comportamento do sistema, a construção de uma linha de base a partir de tráfego benigno e o uso de modelos Autoencoder para identificar desvios em janelas temporais. A partir dessa estrutura, busca-se avaliar como diferentes tipos de anomalia se manifestam em cada nível de observação e em que medida essas perspectivas contribuem para uma análise mais completa do comportamento do sistema.

1.1 Local do Estágio

O estágio foi realizado no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded), vinculado ao Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande. O laboratório atua no desenvolvimento de soluções tecnológicas por meio de projetos de pesquisa, desenvolvimento e inovação, com foco na concepção, implementação e experimentação de sistemas embarcados.

O ambiente do laboratório conta com infraestrutura adequada para atividades práticas, incluindo equipamentos e recursos voltados ao desenvolvimento e teste de aplicações em robótica e automação, possibilitando a realização de experimentos e a validação de soluções em um contexto de pesquisa aplicada.

1.2 Objetivos

O objetivo deste trabalho é desenvolver uma prova de conceito de detecção de anomalias baseada na modelagem do comportamento benigno de um sistema distribuído, utilizando informações coletadas em nível de rede e de aplicação. Para isso, busca-se

definir métricas adequadas para representar o funcionamento do sistema, construir um ambiente experimental controlado, treinar modelos capazes de reconhecer padrões normais de operação e analisar sua capacidade de identificar desvios em cenários distintos de anomalia.

1.3 Organização do Trabalho

O presente relatório está estruturado em cinco capítulos. O Capítulo 1 apresenta a introdução e os objetivos do trabalho. O Capítulo 2 reúne os conceitos e definições fundamentais que dão suporte à proposta. O Capítulo 3 descreve a metodologia adotada e os principais aspectos de implementação. O Capítulo 4 apresenta os resultados obtidos a partir dos experimentos e sua análise. Por fim, o Capítulo 5 traz as considerações finais, com uma síntese dos principais pontos e encaminhamentos do trabalho.

2 Conceitos e Definições

Neste capítulo são apresentados os conceitos fundamentais necessários para a compreensão do trabalho. São introduzidos elementos relacionados à segurança de sistemas, ao monitoramento de tráfego e à detecção de anomalias, além da base conceitual dos modelos utilizados. Esses conceitos servem como suporte para a metodologia adotada, permitindo contextualizar as decisões de modelagem e as abordagens empregadas nas etapas seguintes.

2.1 Segurança de Sistemas e Ameaças

A segurança de sistemas computacionais está relacionada à proteção de serviços e recursos contra acessos indevidos, uso abusivo e interrupções. Em ambientes conectados em rede, o comportamento do sistema é continuamente influenciado por interações externas, o que o torna suscetível a diferentes tipos de ameaça. Essas ameaças podem se manifestar tanto pela exploração de falhas quanto pelo uso indevido de funcionalidades legítimas, refletindo-se em padrões de operação que se afastam do comportamento esperado.

Para interpretar esses desvios, é importante diferenciar alguns conceitos. Vulnerabilidades correspondem a fraquezas do sistema que podem ser exploradas, enquanto ataques são as ações que exploram essas condições ([National Institute of Standards and Technology \(NIST\), 2026](#)). Neste trabalho, essa ideia é representada, por exemplo, pela presença deliberada de uma fraqueza de código no sistema, associada a um padrão conhecido (CWE), que permite a simulação de interações maliciosas. Já o conceito de anomalia refere-se a desvios em relação ao comportamento esperado ([National Institute of Standards and Technology \(NIST\), 2007](#)), independentemente de estarem diretamente ligados a uma exploração específica.

Sistemas expostos em rede possuem uma superfície de ataque composta por elementos acessíveis, como portas e endpoints. Atividades maliciosas tendem a produzir padrões distintos do uso normal. Nesse contexto, a análise do comportamento ao longo do tempo serve de base para as abordagens de monitoramento e detecção de anomalias, que serão detalhadas nas próximas seções.

2.2 Monitoramento de Tráfego

O monitoramento de tráfego consiste na observação contínua das comunicações realizadas em um sistema, com o objetivo de compreender como seus componentes interagem

ao longo do tempo. A partir dessa observação, é possível extrair informações sobre volume de comunicação, frequência de interações e características dos dados trafegados, permitindo a construção de uma visão comportamental do sistema em operação.

Essa análise não se limita a eventos isolados, mas considera a evolução das interações ao longo do tempo, o que permite identificar padrões de funcionamento e possíveis desvios. Alterações na frequência, repetição ou estrutura das comunicações podem indicar mudanças relevantes no comportamento observado, tornando o monitoramento de tráfego uma base importante para técnicas de análise e detecção de anomalias.

2.3 Detecção de Anomalias

A detecção de anomalias consiste na identificação de padrões de comportamento que se desviam do funcionamento esperado de um sistema ([National Institute of Standards and Technology \(NIST\), 2007](#)). Em vez de depender exclusivamente do conhecimento prévio sobre ataques específicos, essa abordagem busca modelar o comportamento considerado normal e, a partir disso, identificar variações significativas que possam indicar situações incomuns ou potencialmente maliciosas.

Essa estratégia é particularmente útil em cenários onde o comportamento pode variar ao longo do tempo ou onde não é possível antecipar todas as formas de ataque. Ao analisar a evolução das interações e comparar novos dados com padrões previamente observados, torna-se possível identificar desvios relevantes, mesmo quando eles não correspondem diretamente a assinaturas conhecidas. Dessa forma, a detecção de anomalias atua como um mecanismo complementar às abordagens tradicionais, permitindo uma análise mais adaptativa do comportamento do sistema.

2.4 Autoencoders

Autoencoders são modelos que aprendem a reconstruir os dados de entrada a partir de uma representação interna compacta ([Keras Documentation, 2026](#)), sendo comumente utilizados em contextos de aprendizado não supervisionado. Sua arquitetura é composta por duas partes principais: um codificador, responsável por mapear a entrada para um espaço latente de menor dimensão, e um decodificador, que reconstrói a entrada original a partir dessa representação. Durante o treinamento, ajusta-se os parâmetros do modelo para minimizar o erro de reconstrução, que aprende a capturar os padrões mais relevantes presentes nos dados.

Uma característica importante desse tipo de modelo é sua sensibilidade a padrões que diferem daqueles observados durante o treinamento. Como o autoencoder aprende a reconstruir apenas o comportamento predominante nos dados fornecidos, entradas que fogem

desse padrão tendem a ser reconstruídas com maior erro. Esse erro de reconstrução pode, portanto, ser utilizado como uma medida de desvio, permitindo identificar comportamentos incomuns sem a necessidade de regras explícitas ou assinaturas pré-definidas.

Outra vantagem relevante é que o treinamento pode ser realizado exclusivamente com dados representativos do comportamento normal do sistema. Isso elimina a necessidade de coletar e rotular exemplos de anomalias, o que muitas vezes é difícil ou inviável na prática. Dessa forma, o modelo aprende uma noção implícita de normalidade, e qualquer desvio significativo em relação a esse padrão pode ser tratado como potencial anomalia, tornando essa abordagem adequada para cenários em que os comportamentos maliciosos são variados ou desconhecidos.

3 Atividades Realizadas

Este capítulo apresenta a metodologia adotada e os detalhes de implementação do ambiente e pipeline propostos para detecção de anomalias. São descritos o ambiente experimental desenvolvido, as estratégias de coleta de dados em nível de rede e aplicação, e o processo de construção do dataset utilizado para treinamento dos modelos. A abordagem é baseada na modelagem do comportamento benigno do sistema, a partir do qual são derivados padrões de referência para identificação de desvios.

A partir dessa base, são apresentados os procedimentos de treinamento dos modelos de detecção, bem como o pipeline de monitoramento e inferência em tempo real, responsável por aplicar os modelos sobre dados observados continuamente. Por fim, são descritos os cenários de anomalia considerados, utilizados para avaliar a capacidade do sistema em identificar diferentes tipos de comportamento anômalo.

3.1 Criação do Ambiente Experimental

Para viabilizar a análise de comportamento e a detecção de anomalias, foi desenvolvido um ambiente experimental baseado em um sistema de automação de dispositivos distribuídos. Esse sistema tem como objetivo simular interações entre diferentes componentes, permitindo a geração controlada de tráfego, tanto em condições normais quanto em cenários de comportamento anômalo, de forma reproduzível. A arquitetura adotada sustenta as etapas de definição das métricas, coleta de dados e execução dos mecanismos de detecção.

A implementação experimental foi executada em uma Raspberry Pi, utilizada como plataforma dedicada para a execução do sistema e dos mecanismos de monitoramento. Essa escolha permitiu observar o tráfego gerado em um ambiente mais isolado, com menor influência de comunicações externas ao experimento. Além disso, o uso dessa plataforma é coerente com o contexto de aplicações IoT, em que dispositivos de baixo custo podem atuar como base para integração, controle e observação do funcionamento do sistema.

3.1.1 Desenvolvimento do Hub e Dispositivos

O ambiente experimental foi estruturado com base em uma arquitetura composta por um hub central e múltiplos dispositivos simulados. O hub é responsável por concentrar a lógica de controle do sistema, recebendo requisições, processando comandos e coordenando as interações entre os dispositivos. Já os dispositivos representam serviços IoT simulados,

com estado próprio, que expõem funcionalidades específicas e respondem às instruções recebidas.

A comunicação entre o hub e os dispositivos é realizada por meio de requisições em rede, seguindo um modelo cliente-servidor. Cada dispositivo mantém uma interface bem definida para recebimento de comandos e envio de respostas, permitindo que o hub atue como ponto central de orquestração. Essa separação facilita a simulação de diferentes cenários de uso, bem como a geração de padrões distintos de tráfego no sistema.

A implementação foi realizada utilizando serviços independentes, executados como processos separados, de forma a representar uma arquitetura distribuída emulada. Essa abordagem permite controlar o comportamento de cada componente individualmente, possibilitando a reprodução tanto de operações normais quanto de variações que simulam condições anômalas.

3.1.2 Fluxos e Interações

As interações no ambiente experimental são organizadas a partir de fluxos de comunicação entre cliente, hub e dispositivos, representando operações típicas de um sistema IoT. Nesse modelo, o cliente envia requisições ao hub, que atua como ponto central de controle, sendo responsável por interpretar os comandos recebidos, incluindo mecanismos de autenticação/autorização, e encaminhar as ações necessárias aos dispositivos envolvidos.

Os fluxos implementados abrangem um conjunto variado de operações, incluindo acionamento de dispositivos (como ligar ou desligar luzes), controle de acesso (travar e destravar fechaduras), ajuste de parâmetros contínuos (como temperatura em termostatos), além de consultas de estado e interações sucessivas entre múltiplos componentes. Além disso, os dispositivos também podem enviar eventos ao hub, reportando mudanças de estado ou ocorrências internas, o que contribui para a dinâmica de comunicação observada no sistema. Essa variedade de funcionalidades permite representar diferentes padrões de uso do sistema, desde interações simples até sequências de ações com maior volume e frequência de comunicação.

```
janfilho@raspberrypi:~ $ curl -s -X POST "$HUB_URL/command" \
-H "Content-Type: application/json" \
-H "X-API-Key: $HUB_API_KEY" \
-d '{"device_id":"light_1","action":"turn_on","request_id":"req-001"}'
{"ok":true,"device_id":"light_1","action":"turn_on","forwarded_to":"http://localhost:8001/command",
"device_response":{"ok":true,"device_type":"light","device_id":"light_1","applied_action":"turn_on",
"new_state":{"state":"on"},"processing_ms":0.01},"hub_latency_ms":24.36}janfilho@raspberrypi:~ $
```

Figura 1 – Envio de um comando ao sistema para acionar uma lâmpada, com resposta do hub após o encaminhamento ao dispositivo correspondente

As interações são realizadas por meio de requisições HTTP, com payloads estruturados no formato JSON, como ilustrado na Figura 1. O hub expõe múltiplos endpoints com responsabilidades distintas, responsáveis por receber comandos, consultar estados e

encaminhar solicitações aos dispositivos. Além disso, o hub também recebe eventos provenientes dos dispositivos e mantém registros de telemetria e logs de aplicação, o que contribui para a observação do comportamento do sistema durante a execução dos experimentos. Esses fluxos não apenas viabilizam a operação do sistema, mas também são fundamentais para a geração de padrões de comunicação observáveis, que são posteriormente utilizados nas etapas de análise e detecção de anomalias.

3.2 Definição de Métricas

Nesta etapa, foram definidas as métricas utilizadas para representar o comportamento do sistema. Essa definição foi realizada em conjunto com a preparação do processo de coleta, com o objetivo de garantir que as informações obtidas sejam relevantes para a análise, permitindo capturar padrões de funcionamento e possíveis desvios ao longo do tempo.

Para isso, o sistema foi analisado sob duas perspectivas complementares: nível de rede e nível de aplicação. No nível de rede, são consideradas informações extraídas a partir de fluxos de comunicação agregados, permitindo uma visão mais generalista do comportamento do tráfego, com menor dependência da semântica da aplicação. Já no nível de aplicação, são observadas as interações registradas pelo hub, por meio de logs e telemetria das requisições, possibilitando uma análise mais específica das operações realizadas. A combinação dessas duas abordagens permite capturar diferentes aspectos do comportamento do sistema, aumentando a capacidade de identificar desvios em relação ao padrão esperado.

3.2.1 Métricas em Nível de Rede

No nível de rede, são consideradas informações extraídas a partir dos fluxos de comunicação observados no sistema. Esses fluxos representam agregações de pacotes trocados entre duas entidades ao longo da comunicação, permitindo capturar características relevantes do comportamento do tráfego com menor dependência da semântica da aplicação.

A partir desses fluxos, são extraídas métricas relacionadas a volume, duração e direção da comunicação. Os principais dados considerados são:

- `dst_port`: porta de destino associada ao fluxo
- `application_name`: classificação inferida para o fluxo
- `bidirectional_packets`: número total de pacotes no fluxo
- `bidirectional_bytes`: volume total de dados trafegados

- `bidirectional_duration_ms`: duração total da comunicação
- `src2dst_packets`: número de pacotes no sentido origem-destino
- `src2dst_bytes`: volume de dados no sentido origem-destino
- `src2dst_duration_ms`: tempo observado no sentido origem-destino
- `dst2src_packets`: número de pacotes no sentido destino-origem
- `dst2src_bytes`: volume de dados no sentido destino-origem
- `dst2src_duration_ms`: tempo observado no sentido destino-origem

A separação entre métricas bidirecionais e por direção permite capturar possíveis assimetrias no padrão de comunicação, que podem ser relevantes para a identificação de comportamentos anômalos.

Neste momento, são considerados apenas os dados brutos extraídos dos fluxos, que serão transformados em features por meio das etapas de agregação e processamento descritas nas seções seguintes.

3.2.2 Métricas em Nível de Aplicação

No nível de aplicação, são consideradas informações extraídas a partir das requisições processadas pelo hub, com base nos registros de logs e telemetria do hub, consolidados no arquivo `hub_requests.csv` durante a execução do sistema. Diferentemente do nível de rede, essas métricas refletem diretamente o comportamento das operações realizadas, permitindo uma análise mais próxima da lógica da aplicação.

Os dados considerados correspondem a atributos observáveis em cada requisição, sem aplicação de agregações neste momento. As principais métricas extraídas são:

- `timestamp_ms`: instante em que a requisição foi registrada
- `is_error`: indica se a requisição resultou em erro
- `is_auth_failure`: falha de autenticação associada ao uso da chave de API
- `path`: endpoint acessado na requisição
- `query_length`: tamanho da query associada à requisição
- `query_entropy`: entropia da string de query

Essas informações permitem observar padrões de uso do sistema em nível mais detalhado, incluindo frequência de acesso a endpoints, ocorrência de erros e características do conteúdo das requisições. Neste momento, são considerados apenas os dados brutos extraídos das requisições, que serão transformados em features por meio das etapas de agregação descritas nas seções seguintes. Essas métricas complementam as informações de rede ao incorporar contexto sobre o tipo de operação realizada e o comportamento do cliente.

3.3 Modelagem do Comportamento Base

Nesta etapa, foi definida a forma como o comportamento normal do sistema é representado para fins de treinamento dos modelos de detecção. Como a abordagem adotada se baseia em aprendizado a partir de padrões de funcionamento esperados, é necessário estabelecer um conjunto de interações que caracterizem o uso legítimo do sistema. Para isso, foi construída uma estratégia de simulação de comportamento, responsável por gerar tráfego benigno de forma controlada, servindo como base para as etapas subsequentes do trabalho.

3.3.1 Definição de Comportamento Base

O comportamento normal do sistema foi definido a partir da simulação de interações típicas entre cliente, hub e dispositivos, com o objetivo de representar diferentes níveis de atividade observados em um ambiente de uso legítimo. Para isso, foi implementado um gerador de requisições responsável por emular o comportamento de um cliente ao longo do tempo, produzindo tráfego benigno de forma controlada e reproduzível.

A geração das interações é baseada em quatro modos de operação: *idle*, *sparse*, *normal* e *busy*, que representam diferentes níveis de atividade do sistema. A seleção do modo ativo é realizada por amostragem probabilística, com pesos definidos para cada modo, sendo *idle* (0,20), *sparse* (0,25), *normal* (0,45) e *busy* (0,10). Essa alternância dinâmica entre modos permite simular variações naturais de uso ao longo do tempo.

Cada modo define uma faixa de duração durante a qual permanece ativo, bem como o intervalo entre o envio de requisições ao hub. No modo *idle*, a duração varia entre 12 e 45 segundos, sem envio de requisições. No modo *sparse*, a duração varia entre 20 e 90 segundos, e as requisições são enviadas com intervalos entre 8 e 25 segundos. No modo *normal*, a duração varia entre 30 e 180 segundos, com intervalos entre requisições de 1,5 a 4,0 segundos. Já no modo *busy*, a duração varia entre 8 e 30 segundos, com intervalos entre requisições de 0,2 a 1,2 segundos. Esses parâmetros controlam diretamente a frequência e a intensidade das interações geradas.

Dentro de cada modo, os endpoints são selecionados por meio de uma distribuição ponderada fixa, refletindo a frequência de uso de cada operação no sistema. Os pesos adotados foram:

- `/command`: 0,38
- `/state`: 0,24
- `/health`: 0,12
- `/event`: 0,08
- `/events`: 0,06
- `/firmware`: 0,02
- `/demo/search`: 0,07
- `/demo/upload-preview`: 0,03

Essa distribuição reflete um cenário em que determinadas operações são mais frequentes do que outras, caracterizando um padrão de uso não uniforme.

Cada tipo de operação possui uma lógica específica de geração, de acordo com o endpoint selecionado. No caso de operações de controle (`/command`), é escolhido um dispositivo aleatório e uma ação compatível com seu tipo, como ligar, desligar, travar, destravar ou ajustar temperatura. Para consultas (`/state`), são selecionados dispositivos aleatórios para obtenção de seu estado atual. No caso de `/event`, são geradas requisições de envio de eventos ao hub, coerentes com o tipo do dispositivo, e a consulta de eventos (`/events`) utiliza um parâmetro `limit` definido de forma ponderada, controlando a quantidade de registros retornados.

Além das operações centrais do sistema, também são simuladas interações auxiliares. O endpoint `/firmware` realiza uploads multipart com conteúdo binário de pequeno tamanho, enquanto `/demo/search` executa buscas a partir de um conjunto pré-definido de consultas e `/demo/upload-preview` simula envios multipart com campos textuais e, em alguns casos, anexos. Essas operações contribuem para a diversificação do tráfego gerado, incorporando padrões adicionais de comunicação ao comportamento considerado normal.

Antes do início do ciclo principal de geração, é realizado um *health check* inicial, garantindo que o sistema esteja disponível para receber requisições. A geração das interações é conduzida por um cliente HTTP assíncrono, com duração total configurável e controle de timeout para as requisições. Além disso, o processo utiliza uma semente de aleatoriedade configurável, permitindo a reprodução dos experimentos.

O comportamento gerado não corresponde a uma sequência fixa de comandos, mas sim a um processo dinâmico, no qual os modos de operação alternam ao longo do tempo e, dentro de cada modo, endpoints e parâmetros são selecionados de forma aleatória ponderada. Dessa forma, o tráfego benigno gerado incorpora variabilidade suficiente para representar diferentes padrões de uso, incluindo tanto operações típicas de dispositivos IoT quanto interações com endpoints auxiliares e de demonstração.

3.3.2 Rotinas para Geração de Tráfego Benigno

A partir da definição do comportamento base, foi realizada a geração de tráfego benigno com o objetivo de simular a operação do sistema em condições normais de uso. Essa geração foi executada durante um período de quatro horas, utilizando o gerador de requisições descrito anteriormente, com alternância dinâmica entre os modos de operação e seleção probabilística de endpoints e parâmetros. Essa execução serviu como base controlada para a coleta do comportamento benigno utilizado posteriormente no experimento.

É importante destacar que o tráfego gerado corresponde a uma simulação controlada de comportamento benigno. Em um ambiente real, a linha de base de comportamento seria obtida por meio do monitoramento contínuo do sistema em produção, ao longo de um período representativo de operação.

3.4 Coleta de Dados

A coleta de dados teve como objetivo registrar os dados brutos necessários para as demais etapas do pipeline. Essa coleta foi realizada em dois níveis complementares, rede e aplicação, permitindo observar o comportamento do sistema sob diferentes perspectivas. As estratégias adotadas foram utilizadas tanto na execução controlada com geração de tráfego benigno quanto no monitoramento contínuo do sistema em tempo real, posteriormente.

3.4.1 Coleta em Nível de Rede

A coleta em nível de rede consiste na observação do tráfego na interface monitorada do sistema, sendo realizada diretamente no ambiente experimental onde a aplicação, os dispositivos simulados e o gerador de tráfego são executados. No cenário adotado, o monitoramento ocorre na interface `lo`, uma vez que toda a comunicação se dá localmente. Para essa coleta, é utilizada a biblioteca `NFStream` ([NFStream Project, 2026](#)), que realiza o monitoramento direto do tráfego e sua conversão em registros de fluxo, definidos como agregações de pacotes observados entre duas entidades durante a comunicação.

O resultado dessa coleta é um conjunto de registros em que cada linha representa um fluxo observado ainda em nível bruto. Os campos presentes nesses registros correspondem às

informações utilizadas posteriormente na etapa de construção de *features*. Essa abordagem permite capturar o comportamento do tráfego, e serve tanto para a etapa de simulação de comportamento benigno quanto para o monitoramento contínuo do sistema em tempo de execução.

3.4.2 Coleta em Nível de Aplicação

A coleta em nível de aplicação é realizada a partir do hub, que atua como ponto central de observação do sistema. Os dados são obtidos a partir dos logs e da telemetria gerados durante o processamento das requisições, sendo registrados automaticamente ao longo da execução. Essa coleta ocorre de forma contínua, com registro individual por requisição, e os dados são consolidados no arquivo `hub_requests.csv`.

O resultado é um conjunto de dados em que cada linha representa uma requisição processada pelo hub. Essa coleta fornece uma visão mais próxima da lógica da aplicação, incorporando informações semânticas sobre as operações realizadas, e atua como complemento ao nível de rede. Os dados obtidos são utilizados tanto na execução controlada com tráfego benigno quanto no monitoramento contínuo do sistema, sendo utilizados nas etapas de agregação temporal e derivação de *features*.

3.5 Construção do Dataset

A construção do dataset tem como objetivo transformar os dados brutos coletados em uma representação estruturada adequada para treino dos modelos e inferência. A partir das métricas definidas na seção 3.2 e dos dados obtidos nos níveis de rede e aplicação durante a execução com tráfego benigno, foram derivados atributos que representam o comportamento do sistema ao longo do tempo. Essa etapa estabelece a unidade de análise adotada e define como os dados coletados são convertidos em vetores de *features* utilizados nas etapas seguintes.

3.5.1 Agrupamento Temporal

O comportamento do sistema não é definido por eventos isolados, mas pela forma como as interações ocorrem ao longo do tempo. Métricas calculadas a partir de eventos individuais tendem a perder contexto e não capturam adequadamente variações de carga, padrões de uso ou mudanças no comportamento. Por esse motivo, a análise foi realizada sobre intervalos de tempo.

Para isso, o tempo foi dividido em janelas fixas de 10 segundos. Cada janela agrupa os dados observados nos dois níveis de coleta, fluxos no nível de rede e requisições no nível de aplicação. A unidade de análise passa a ser a janela temporal, e não os eventos

individuais. A referência temporal difere entre os níveis: no nível de aplicação, cada requisição foi associada à janela com base em seu `timestamp_ms`, enquanto no nível de rede os fluxos são associados com base no instante final observado.

No nível de rede, os fluxos só são disponibilizados após o término da comunicação, de modo que seu instante de início não representa o momento em que o dado se torna observável. Para lidar com isso, foi utilizado o campo `bidirectional_last_seen_ms` como referência temporal, de modo que cada fluxo é associado à janela correspondente ao seu instante final, isto é, ao momento em que se torna efetivamente observável no monitoramento. Essa decisão permite construir o dataset de forma coerente com a dinâmica real de observação dos fluxos, preservando a referência temporal utilizada nas etapas posteriores do trabalho.

3.5.2 Construção das Features

A construção das features para treinamento dos Autoencoders foi realizada a partir dos dados brutos coletados nos níveis de rede e aplicação, após o agrupamento temporal. Esses dados correspondem, respectivamente, aos fluxos de comunicação e às requisições processadas pelo hub. Nesta etapa, são construídos dois datasets independentes, um em nível de rede (*flow-level*) e outro em nível de aplicação (*app-level*), sem fusão entre os dois níveis.

No nível de aplicação, as features são derivadas de agregações sobre as requisições registradas pelo hub em cada janela:

- `requests_total`: número total de requisições na janela
- `error_rate`: proporção de requisições que resultaram em erro
- `auth_failure_rate`: proporção de falhas de autenticação
- `unique_path_ratio`: diversidade de endpoints acessados
- `avg_query_length`: tamanho médio das queries nas requisições
- `entropy_mean`: entropia média das queries
- `entropy_max`: maior valor de entropia observado nas queries

No nível de rede, as features são obtidas a partir da agregação dos fluxos dentro de cada janela:

- `n_communication`: quantidade de fluxos observados na janela
- `dst_port_entropy`: diversidade de portas de destino

- `app_entropy`: diversidade das classificações de aplicação

Além disso, são calculadas estatísticas para `packets`, `bytes` e `duration`, nas visões `bidirectional`, `src2dst` e `dst2src`, a partir das métricas originais:

- totais (ex.: `bidirectional_bytes_total`): agregação de pacotes, bytes e duração
- médias (ex.: `bidirectional_packets_mean`): valores médios por fluxo na janela
- máximos (ex.: `src2dst_packets_max`): maiores valores observados entre os fluxos

Essas métricas capturam padrões de comunicação em termos de distribuição, volume e assimetria do tráfego. Como resultado, cada janela temporal é representada por um vetor de features, definindo a unidade final de análise. São gerados dois datasets estruturados, um para cada nível, utilizados posteriormente para o treinamento de dois modelos, um para cada perspectiva de monitoramento.

3.6 Treinamento dos Modelos

O treinamento foi realizado de forma independente para os datasets *flow-level* e *app-level*, resultando em dois modelos distintos de detecção. Em ambos os casos, foi adotada uma abordagem baseada em Autoencoders treinados exclusivamente com dados benignos, com o objetivo de aprender padrões de funcionamento normal do sistema e utilizá-los como referência para a identificação de desvios.

Antes do treinamento, os vetores de features foram normalizados com base nos dados de treino, e essa mesma transformação foi mantida nas etapas de validação e inferência. Para o dataset em nível de aplicação, foi utilizada uma divisão de 80% para treino e 20% para validação. No dataset em nível de rede, a divisão adotada foi de 90% para treino e 10% para validação.

O treinamento foi conduzido por 50 épocas, utilizando como critério de ajuste o erro quadrático médio (MSE) entre a entrada e sua reconstrução. Em ambos os modelos, a arquitetura segue a lógica de codificação, projeção em um espaço latente e reconstrução da entrada, como ilustrado na Figura 2. No modelo em nível de rede, foi utilizado um espaço latente de dimensão 4, enquanto no modelo em nível de aplicação foi adotada dimensão 3. A escolha de um espaço latente maior para o nível de rede está associada ao fato de esse modelo operar sobre um conjunto maior de features.

Após o treinamento, o erro de reconstrução passou a ser utilizado como medida de desvio em relação ao comportamento benigno aprendido. Para cada modelo, foi definido um limiar de decisão a partir dos erros observados no conjunto de validação benigno,

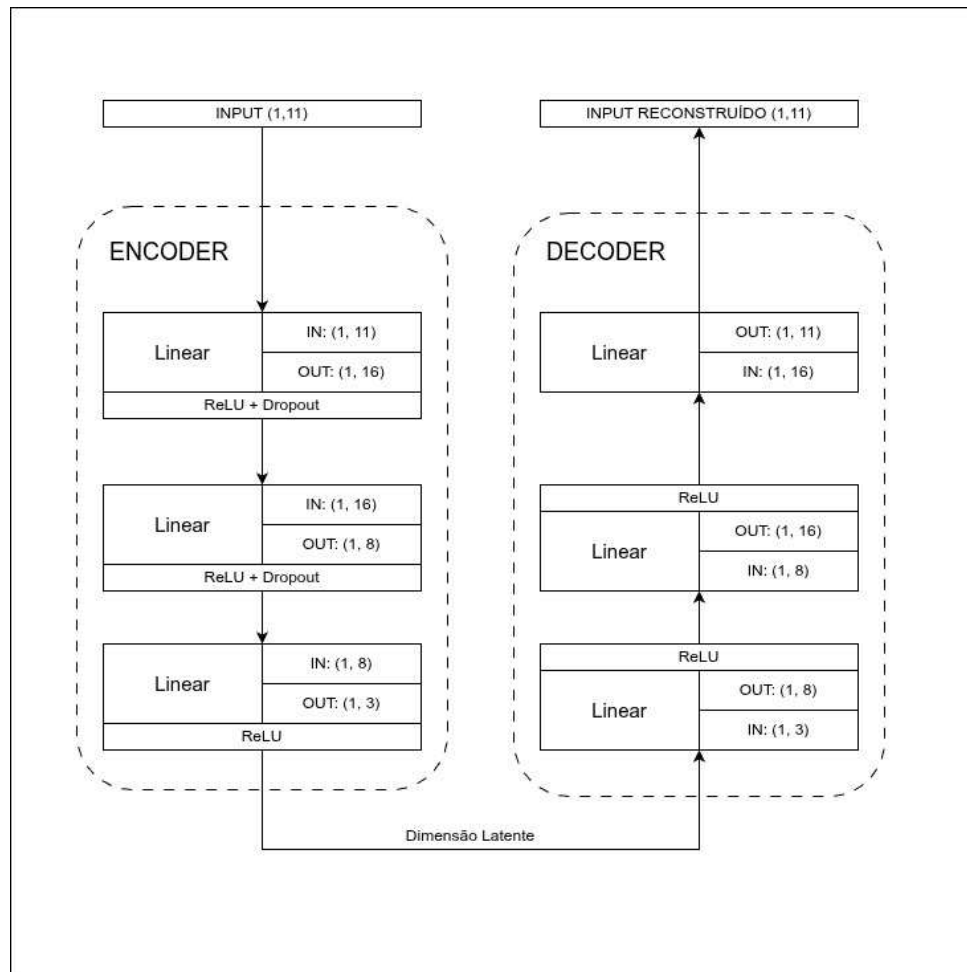


Figura 2 – Esquema geral da arquitetura do Autoencoder para o nível de rede

permitindo classificar novas janelas como benignas ou anômalas. Como resultado, foram obtidos dois modelos independentes: um voltado ao monitoramento em nível de rede e outro ao monitoramento em nível de aplicação.

3.7 Monitoramento e Inferência em Tempo Real

Após a construção do dataset e o treinamento dos modelos, inicia-se a etapa de monitoramento em tempo real do sistema. Nesse cenário, o hub e os dispositivos permanecem em execução normal, enquanto os mecanismos de coleta em nível de rede e aplicação atuam em paralelo, observando continuamente os dados gerados durante o funcionamento do sistema.

Esse monitoramento utiliza a mesma lógica definida na etapa de construção do dataset: os dados são coletados continuamente, organizados em janelas temporais de 10 segundos e transformados em features. Cada janela representa uma unidade de análise em tempo real, sendo utilizada como entrada para os modelos treinados, visando a detecção de desvios de comportamento.

3.7.1 Pipeline de Monitoramento

O monitoramento é realizado por processos que executam em paralelo ao funcionamento do sistema, observando continuamente os dados gerados pelo hub e pelos dispositivos sem interferir na operação. Esses processos são responsáveis por transformar o fluxo de dados em janelas temporais utilizadas como entrada para a inferência.

No nível de rede, o monitor observa o tráfego na interface utilizada pelo sistema, registrando os fluxos em arquivos CSV. Além de um arquivo acumulado, são gerados arquivos por janela temporal, nos quais os fluxos são agrupados com base no campo `bidirectional_last_seen_ms`. No nível de aplicação, o hub registra as requisições em `hub_requests.csv`, e um processo de monitoramento lê esse arquivo continuamente, organizando as requisições em janelas com base no campo `timestamp_ms`.

Os dados são organizados em janelas fixas de 10 segundos, sendo gerados arquivos no formato `window_<start_ms>_<end_ms>.csv`, contendo os dados brutos observados em cada intervalo. O resultado do monitoramento é a produção contínua dessas janelas temporais, que servem de base para a etapa de inferência.

3.7.2 Inferência Contínua

A inferência contínua é realizada por processos independentes, um para *flow-level* e outro para *app-level*, que executam em paralelo ao monitoramento e observam continuamente os diretórios onde as janelas temporais são geradas. A lógica de consumo é incremental: os processos listam os arquivos no formato `window_<start_ms>_<end_ms>.csv` e processam apenas janelas ainda não analisadas, mantendo estado para evitar reprocessamento. Uma janela só é considerada apta para inferência quando seu instante final já foi ultrapassado e um *grace period* adicional de 10 segundos é respeitado, reduzindo a chance de processar janelas que ainda estão recebendo dados.

Para cada janela elegível, os dados brutos são lidos diretamente dos arquivos gerados pelo monitoramento e transformados em um vetor de features por meio da mesma lógica de agregação utilizada na etapa offline. No nível de rede, são utilizadas as features derivadas dos fluxos, enquanto no nível de aplicação são utilizadas as features derivadas das requisições, sem fusão entre os dois níveis. Essa derivação das features em tempo de execução garante consistência entre o ambiente de treinamento e o monitoramento em tempo real.

A inferência consiste, para cada janela, na aplicação do autoencoder correspondente, seguido do cálculo do erro de reconstrução e sua comparação com o limiar previamente definido, resultando em uma classificação binária por janela, indicando comportamento benigno ou anômalo. Os resultados são registrados continuamente em arquivos CSV de saída, e cada janela processada é marcada como já analisada por meio desse histórico.

3.8 Cenários de Anomalia

Após a definição do comportamento benigno e do pipeline de monitoramento, foram considerados cenários de anomalia com o objetivo de avaliar a capacidade do sistema em detectar diferentes padrões de desvio. Esses cenários representam comportamentos distintos, incluindo exploração da superfície de rede, abuso de mecanismos de autenticação e exploração da lógica da aplicação. Dessa forma, busca-se observar como diferentes tipos de anomalia se manifestam nos níveis de rede e aplicação, de maneira complementar.

3.8.1 Network Reconnaissance

O cenário de *Network Reconnaissance* representa atividades de sondagem da superfície de rede exposta pelo sistema, tipicamente realizadas por meio do envio de múltiplas tentativas de conexão a diferentes portas ou serviços. Esse comportamento busca identificar quais serviços estão ativos e acessíveis, variando parâmetros como porta de destino e frequência de conexões em um curto intervalo de tempo.

Esse tipo de atividade é anômalo por introduzir padrões de comunicação que não fazem parte do uso normal da aplicação, caracterizados por aumento na diversidade de portas acessadas e no número de fluxos distintos observados. No contexto deste trabalho, os desvios tendem a se manifestar principalmente no nível de rede, afetando métricas como entropia de portas de destino, quantidade de comunicações e distribuição de fluxos, sendo pouco perceptíveis no nível de aplicação.

3.8.2 Brute Force de API Key

O cenário de *Brute Force de API Key* consiste na realização sistemática de múltiplas requisições autenticadas com chaves inválidas ou aleatórias, com o objetivo de identificar uma credencial válida. Esse comportamento se caracteriza pela repetição de tentativas de acesso ao mesmo endpoint, frequentemente com variações nos valores de autenticação utilizados em cada requisição.

Esse padrão é anômalo por introduzir uma frequência elevada de falhas de autenticação em um curto intervalo de tempo, destoando do comportamento esperado de clientes legítimos. No sistema analisado, esse cenário tende a impactar diretamente o nível de aplicação, aumentando métricas como `auth_failure_rate` e `error_rate`, e também pode gerar efeitos no nível de rede devido ao volume e à repetição das comunicações.

3.8.3 Enumeration Attack

O cenário de *Enumeration Attack* representa a exploração sistemática dos endpoints da aplicação por meio do envio de requisições para diferentes rotas, com o objetivo

de identificar funcionalidades disponíveis. Esse comportamento envolve a variação dos caminhos acessados, muitas vezes incluindo rotas inexistentes ou não documentadas, resultando em respostas de erro ou comportamento inesperado.

Esse tipo de atividade é anômalo por alterar significativamente o padrão de acesso à aplicação, aumentando a diversidade de endpoints utilizados em um curto período. No contexto do monitoramento, esse desvio se manifesta principalmente no nível de aplicação, afetando métricas como `unique_path_ratio` e taxas de erro, com possível impacto também nos fluxos observados no nível de rede.

3.8.4 SSTI (CWE-1336) / RCE

Este cenário foi introduzido deliberadamente no ambiente experimental por meio de um endpoint de busca mantido propositalmente com uma fraqueza de injeção em templates do lado do servidor (*Server-Side Template Injection*) (CWE-1336, 2026). Em termos práticos, esse tipo de problema ocorre quando um valor fornecido pelo usuário, que deveria ser tratado apenas como texto (por exemplo, um termo de busca), passa a ser visto pelo servidor como uma expressão a ser interpretada. Isso faz com que a entrada deixe de ser apenas um dado e passe a influenciar diretamente o processamento da aplicação.

No sistema implementado, essa condição foi mantida no endpoint de busca para permitir a simulação de entradas especialmente construídas que alteram o comportamento esperado da aplicação. Em cenários mais graves, esse tipo de exploração pode levar à execução indevida de expressões ou comandos no servidor, sendo associado a *Remote Code Execution* (CWE-94, 2026). No contexto deste trabalho, o objetivo é observar os padrões de desvio gerados por esse tipo de interação, que tendem a se manifestar principalmente no nível de aplicação, por meio de alterações principalmente no tamanho e na entropia das *queries*.

Em conjunto, esses cenários permitem observar como diferentes tipos de desvio se manifestam no sistema, evidenciando padrões distintos tanto no nível de rede quanto no nível de aplicação. Essa variedade de comportamentos anômalos fornece uma base inicial para avaliar a capacidade do monitoramento em identificar mudanças relevantes no funcionamento do sistema.

4 Resultados

Este capítulo apresenta os resultados obtidos a partir da execução dos experimentos definidos ao longo do trabalho. Inicialmente, é feita a caracterização do comportamento benigno utilizado como referência. Em seguida, são apresentados os resultados de detecção de anomalias para os diferentes cenários considerados. Por fim, são discutidas as implicações desses resultados, com base no comportamento observado nos níveis de rede e aplicação.

4.1 Caracterização do Comportamento Base

Ao longo da execução da coleta dos dados em nível de rede e aplicação durante quatro horas, foram registrados 4451 eventos em `hub_requests.csv` e 7869 fluxos em `flows_full.csv`, distribuídos em 1345 janelas temporais não vazias de 10 segundos. Esses valores mostram que o comportamento base observado corresponde a um processo com variação temporal e volume suficiente para representar diferentes padrões de operação do sistema.

A diferença entre o número de requisições e o número de fluxos é compatível com a arquitetura do ambiente experimental, na qual uma mesma interação pode produzir múltiplos fluxos observáveis entre os dispositivos simulados. Dessa forma, os dois níveis de monitoramento oferecem descrições complementares do comportamento benigno: o nível de aplicação registra as operações processadas pelo hub, enquanto o nível de rede captura a dinâmica de comunicação resultante dessas interações.

Os testes manuais realizados no ambiente de monitoramento em tempo de execução indicaram que interações benignas não produziram sinalizações relevantes de anomalia, em contraste com os cenários anômalos executados posteriormente. Em conjunto, esses resultados reforçam a adequação da linha de base adotada como referência.

4.2 Detecção de Anomalias

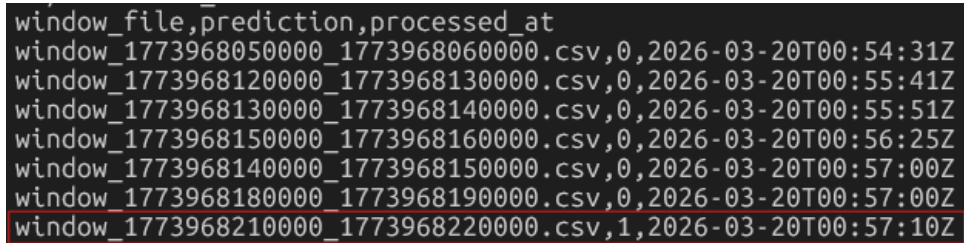
Nesta seção são apresentados os resultados da detecção de anomalias para os cenários definidos anteriormente. Para cada caso, descreve-se o comportamento executado no experimento, as evidências observadas no monitoramento e como essas evidências se refletem nas métricas utilizadas pelos modelos. A análise é conduzida separando os efeitos em nível de rede e de aplicação, destacando em quais situações cada abordagem foi mais eficaz na identificação dos desvios em relação ao comportamento benigno.

4.2.1 Network Reconnaissance

O cenário de *Network Reconnaissance* foi executado por meio de uma varredura repetitiva de portas contra o próprio dispositivo que executa o hub, utilizando o seguinte comando:

```
1 timeout 60 bash -c 'while true; do nmap -sT -Pn -n -p 8000-9000 127.0.0.1 >/dev
  /null; done'
```

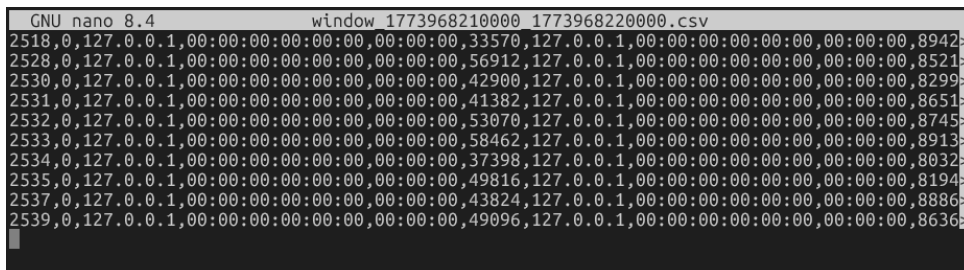
No contexto do experimento, esse comando realiza varreduras TCP conectivas (*-sT*) de forma contínua, desabilitando descoberta por *ping* (*-Pn*) e resolução de nomes (*-n*), testando repetidamente a faixa de portas de 8000 a 9000 no próprio sistema local (127.0.0.1). Na prática, isso gera uma sequência intensa de tentativas de conexão de curta duração, distribuídas ao longo de múltiplas portas.



```
window_file,prediction,processed_at
window_1773968050000_1773968060000.csv,0,2026-03-20T00:54:31Z
window_1773968120000_1773968130000.csv,0,2026-03-20T00:55:41Z
window_1773968130000_1773968140000.csv,0,2026-03-20T00:55:51Z
window_1773968150000_1773968160000.csv,0,2026-03-20T00:56:25Z
window_1773968140000_1773968150000.csv,0,2026-03-20T00:57:00Z
window_1773968180000_1773968190000.csv,0,2026-03-20T00:57:00Z
window_1773968210000_1773968220000.csv,1,2026-03-20T00:57:10Z
```

Figura 3 – Anomalia detectada em janela de 10s durante a varredura do Nmap

Durante a execução, as janelas temporais de 10 segundos que cobriram períodos de *scan* foram classificadas como anômalas no nível de rede, como o exemplo na Figura 3. Esse comportamento foi identificado principalmente devido ao aumento expressivo da diversidade de portas de destino, refletido na métrica *dst_port_entropy*, e ao crescimento no número de fluxos observados por janela (como ilustrado na Figura 4), capturado por *n_communication*. Além disso, foram observados muitos fluxos de duração muito curta, com valores praticamente nulos de *bidirectional_duration_ms*, compatíveis com simples verificações de disponibilidade de portas.



```
GNU nano 8.4 window_1773968210000_1773968220000.csv
2518,0,127.0.0.1,00:00:00:00:00:00,00:00:00,33570,127.0.0.1,00:00:00:00:00:00,00:00:00,8942>
2528,0,127.0.0.1,00:00:00:00:00:00,00:00:00,56912,127.0.0.1,00:00:00:00:00:00,00:00:00,8521>
2530,0,127.0.0.1,00:00:00:00:00:00,00:00:00,42900,127.0.0.1,00:00:00:00:00:00,00:00:00,8299>
2531,0,127.0.0.1,00:00:00:00:00:00,00:00:00,41382,127.0.0.1,00:00:00:00:00:00,00:00:00,8651>
2532,0,127.0.0.1,00:00:00:00:00:00,00:00:00,53070,127.0.0.1,00:00:00:00:00:00,00:00:00,8745>
2533,0,127.0.0.1,00:00:00:00:00:00,00:00:00,58462,127.0.0.1,00:00:00:00:00:00,00:00:00,8913>
2534,0,127.0.0.1,00:00:00:00:00:00,00:00:00,37398,127.0.0.1,00:00:00:00:00:00,00:00:00,8032>
2535,0,127.0.0.1,00:00:00:00:00:00,00:00:00,49816,127.0.0.1,00:00:00:00:00:00,00:00:00,8194>
2537,0,127.0.0.1,00:00:00:00:00:00,00:00:00,43824,127.0.0.1,00:00:00:00:00:00,00:00:00,8886>
2539,0,127.0.0.1,00:00:00:00:00:00,00:00:00,49096,127.0.0.1,00:00:00:00:00:00,00:00:00,8636>
```

Figura 4 – Exemplo de janela de 10s com mais de 2500 fluxos observados durante o reconhecimento de rede com Nmap

Esse padrão difere do comportamento benigno aprendido pelos modelos, no qual o tráfego tende a se concentrar em um conjunto limitado de portas (8000 a 8003) e a

apresentar interações mais estáveis e duradouras. As conexões geradas durante o reconhecimento também apresentam pouca troca efetiva de dados de aplicação, reforçando o desvio observado nas métricas de fluxo.

Por outro lado, esse cenário não produziu anomalias no nível de aplicação, uma vez que não produz interações relevantes no nível da aplicação e não gera registros em `hub_requests.csv`. Dessa forma, o desvio foi observado predominantemente no nível de rede. Esse resultado evidencia que o monitoramento em *flow-level* foi mais adequado para capturar atividades de sondagem e reconhecimento da superfície exposta do sistema.

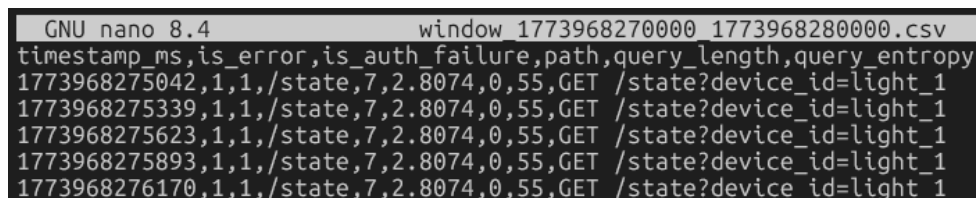
4.2.2 Brute Force de API Key

O cenário de *Brute Force de API Key* foi executado como uma sequência repetitiva de tentativas de acesso com chaves de API inválidas, utilizando uma lista pré-definida de valores. As requisições foram direcionadas ao endpoint de consulta de estado do sistema, conforme o comando:

```
1 timeout 60s bash -lc 'while true; do while read -r key; do curl -s "http://
  localhost:8000/state?device_id=light_1" -H "X-API-Key: $key" >/dev/null;
  sleep 0.2; done < /home/janfilho/net_anomaly_demo_app/runtime/attacks/keys.
  txt; done'
```

Na prática, esse comando envia requisições repetidas ao endpoint `/state`, variando o valor do cabeçalho `X-API-Key` e mantendo esse comportamento continuamente por 60 segundos. Durante a execução, as janelas temporais de 10 segundos foram classificadas como anômalas tanto no nível de aplicação quanto no nível de rede.

No nível de aplicação, o desvio foi observado principalmente pelo aumento acentuado das métricas `auth_failure_rate` e `error_rate`, refletindo a grande quantidade de tentativas de autenticação inválidas concentradas em janelas curtas. A Figura 5 apresenta um exemplo desse comportamento. Além disso, um padrão de acesso altamente repetitivo ao mesmo endpoint também é incompatível com o comportamento benigno aprendido.



timestamp_ms	is_error	is_auth_failure	path	query_length	query_entropy
1773968275042	1	1	/state	7	2.8074
1773968275339	1	1	/state	7	2.8074
1773968275623	1	1	/state	7	2.8074
1773968275893	1	1	/state	7	2.8074
1773968276170	1	1	/state	7	2.8074

Figura 5 – Exemplos de métricas *app-level* para janela de 10s durante o ataque de *Brute Force*

No nível de rede, a detecção ocorreu devido ao volume e à repetição das comunicações em um curto intervalo de tempo, com aumento do número de fluxos observados por janela e concentração do tráfego em uma mesma porta e serviço. Diferentemente do cenário de

reconhecimento de rede, o desvio não está associado à diversidade de portas, mas sim a um padrão intenso e repetitivo de conexões direcionadas ao mesmo destino.

Esse cenário corresponde ao abuso repetitivo do mecanismo de autenticação da aplicação, e produz sinais simultâneos e consistentes nos dois níveis de monitoramento: no nível de aplicação, pelas falhas de autenticação e erros, e no nível de rede, pelo padrão de comunicação repetitiva concentrada em um único serviço.

4.2.3 Enumeration Attack

O cenário de *Enumeration Attack* foi executado por meio do envio sucessivo de requisições para diferentes rotas da aplicação, utilizando uma lista pré-definida de caminhos. O comando utilizado foi:

```
1 while read -r path; do curl -s "http://localhost:8000/$path" >/dev/null; sleep
   0.2; done < /home/janfilho/net_anomaly_demo_app/runtime/attacks/paths.txt
```

Esse comando percorre uma lista de *paths* e envia requisições HTTP para cada um deles, com um pequeno intervalo entre chamadas. No contexto do experimento, esse comportamento simula a enumeração da superfície funcional exposta pelo hub, testando diferentes endpoints, incluindo rotas válidas e inexistentes.

Durante a execução, as janelas temporais de 10 segundos foram classificadas como anômalas no nível de aplicação. O principal desvio foi o acesso a muitos endpoints distintos em um curto intervalo de tempo, elevando a métrica *unique_path_ratio*. Além disso, foram registradas respostas de erro associadas a rotas inexistentes ou não previstas, impactando diretamente as taxas de erro. Na Figura 6 é possível ver a reação do hub durante a execução do ataque. Esse padrão destoou do comportamento benigno aprendido, no qual o acesso tende a se concentrar em um conjunto reduzido de endpoints.

```
INFO: 127.0.0.1:46706 - "GET /system HTTP/1.1" 404 Not Found
{"method":"GET","path":"/systems","status_code":404,"client_ip":"127.0.0.1","latency_ms":1.18,"ts":"2026-03-20T10:43:32.207Z","service":"hub"}
INFO: 127.0.0.1:46722 - "GET /systems HTTP/1.1" 404 Not Found
{"method":"GET","path":"/users","status_code":404,"client_ip":"127.0.0.1","latency_ms":1.03,"ts":"2026-03-20T10:43:32.458Z","service":"hub"}
INFO: 127.0.0.1:46732 - "GET /users HTTP/1.1" 404 Not Found
{"method":"GET","path":"/auth","status_code":404,"client_ip":"127.0.0.1","latency_ms":1.03,"ts":"2026-03-20T10:43:32.709Z","service":"hub"}
INFO: 127.0.0.1:46734 - "GET /auth HTTP/1.1" 404 Not Found
{"method":"GET","path":"/signin","status_code":404,"client_ip":"127.0.0.1","latency_ms":1.2,"ts":"2026-03-20T10:43:32.960Z","service":"hub"}
INFO: 127.0.0.1:46744 - "GET /signin HTTP/1.1" 404 Not Found
{"method":"GET","path":"/signup","status_code":404,"client_ip":"127.0.0.1","latency_ms":1.13,"ts":"2026-03-20T10:43:33.213Z","service":"hub"}
INFO: 127.0.0.1:46756 - "GET /signup HTTP/1.1" 404 Not Found
{"method":"GET","path":"/setup","status_code":404,"client_ip":"127.0.0.1","latency_ms":1.1,"ts":"2026-03-20T10:43:33.465Z","service":"hub"}
INFO: 127.0.0.1:46760 - "GET /setup HTTP/1.1" 404 Not Found
{"method":"GET","path":"/wizard","status_code":404,"client_ip":"127.0.0.1","latency_ms":1.06,"ts":"2026-03-20T10:43:33.717Z","service":"hub"}
INFO: 127.0.0.1:46762 - "GET /wizard HTTP/1.1" 404 Not Found
```

Figura 6 – Registros de auditoria do hub durante tentativas de acesso de endpoints inexistentes

No nível de rede, as janelas também foram classificadas como anômalas. Nesse caso, o desvio está associado ao aumento do número de requisições em um curto intervalo,

resultando em mais fluxos e comunicação repetitiva, de maneira similar ao ataque de *Brute Force*. Diferentemente do reconhecimento de rede, não há variação significativa de portas, mas sim um padrão de requisições intensas.

Esse cenário envolve a tentativa de mapear as funcionalidades expostas pela aplicação. Por esse motivo, ele é capturado principalmente no nível de aplicação, com efeitos secundários observáveis no nível de rede.

4.2.4 SSTI (CWE-1336) / RCE

O cenário de *SSTI* (*Server-Side Template Injection*) foi executado por meio do envio de uma requisição ao endpoint de busca do sistema, utilizando o seguinte comando:

```
1 curl -sG "http://localhost:8000/demo/search" \  
2 --data-urlencode "q={% for c in ''.__class__.__mro__[1].__subclasses__() %}{%  
    if 'Popen' in c.__name__ %}{c('id',shell=True,stdout=-1).communicate()  
    %}{% endif %}{% endfor %}"
```

Esse comando realiza uma requisição HTTP do tipo GET ao endpoint `/demo/search`. O parâmetro `q`, que normalmente receberia um termo simples de busca, é preenchido com uma expressão construída para explorar o mecanismo de template do servidor. A opção `-G` indica que os parâmetros são enviados na URL, enquanto `-data-urlencode` garante que o conteúdo seja codificado corretamente.

No ambiente experimental, esse endpoint foi mantido deliberadamente vulnerável, permitindo que o valor de `q` seja interpretado como uma expressão. Na prática, isso significa que o servidor tenta interpretar o conteúdo recebido, em vez de tratá-lo apenas como texto. O *payload* utilizado explora esse comportamento ao percorrer estruturas internas da linguagem até encontrar a classe responsável por criar novos processos (`Popen`). A partir disso, a expressão solicita a execução do comando `id` no sistema operacional, utilizando a criação de um subprocesso. Esse tipo de construção é comum em cenários de exploração, pois permite transformar uma simples entrada de usuário em execução de comandos no servidor.

Durante a execução, as janelas temporais de 10 segundos foram classificadas como anômalas no nível de aplicação. O desvio foi observado principalmente pelo aumento significativo de `query_length` e `query_entropy`, como pode ser observado na Figura 7, refletindo o envio de entradas muito maiores e estruturalmente mais complexas do que aquelas presentes no comportamento benigno. Esse padrão é incompatível com o uso esperado do endpoint de busca, que, em condições normais, recebe consultas curtas e simples.

Esse cenário difere dos demais por não envolver sondagem de portas, abuso de autenticação ou enumeração de rotas, mas sim a exploração direta de uma entrada

timestamp_ms	is_error	is_auth_failure	path	query_length	query_entropy	request_content_length	response_length	request_preview
1774249070696	0	0	/demo/search	12	3.1887	0	240	GET /demo/search?q=thermostat_1
1774249073007	0	0	/demo/search	154	4.7082	0	504	GET /demo/search?q=%7b%25+for+c+in+%2

Figura 7 – Métricas de comando benigno e malicioso no endpoint `/demo/search`

da aplicação por meio de um *payload*. Nesse caso, o desvio relevante está associado às características da requisição em nível de aplicação, sendo capturado pelas métricas derivadas da query.

Esse resultado evidencia a importância do monitoramento em nível de aplicação, uma vez que o sinal de anomalia está diretamente relacionado às características da entrada processada pelo sistema, não sendo capturado pela análise em nível de rede.

4.3 Análise em Níveis: Rede e Aplicação

A análise dos resultados permite observar como cada cenário de anomalia se manifesta nos diferentes níveis de monitoramento, evidenciando diferenças no tipo de desvio produzido e na forma como ele é capturado pelas métricas definidas. De maneira geral, os cenários não geram sinais homogêneos entre os dois níveis, variando conforme a natureza da interação executada no experimento. A Tabela 1 apresenta um resumo dessas detecções, indicando em quais níveis cada tipo de comportamento foi identificado.

Tabela 1 – Resumo qualitativo da detecção de anomalias por cenário

Cenário	Detecção em Rede	Detecção em Aplicação
Network Reconnaissance	Sim	Não
Brute Force de API Key	Sim	Sim
Enumeration Attack	Sim	Sim
SSTI (CWE-1336) / RCE	Não	Sim

Observa-se que, como esperado, o nível de rede é mais sensível a padrões relacionados à estrutura de comunicação, como volume de fluxos, frequência de conexões e diversidade de portas, sendo mais adequado para capturar atividades como reconhecimento de rede. Por outro lado, o nível de aplicação se mostra mais eficaz na identificação de desvios ligados à lógica das requisições, incluindo erros, falhas de autenticação, variação de endpoints e características do conteúdo enviado. Em cenários como *Brute Force* e *Enumeration*, ambos os níveis contribuem para a detecção, ainda que por motivos distintos. Essa complementaridade justifica a adoção da abordagem em dois níveis, permitindo capturar diferentes dimensões do comportamento do sistema que não seriam plenamente observáveis por apenas uma das perspectivas.

4.4 Discussões

Os resultados obtidos permitem analisar o comportamento da abordagem proposta além dos cenários específicos avaliados, considerando seu alcance, suas limitações estruturais e possibilidades de evolução. A partir das características das métricas utilizadas e dos padrões observados, é possível discutir em que medida o sistema pode ser aplicado a outros contextos e quais aspectos ainda podem ser aprimorados.

A partir das métricas utilizadas, é possível identificar outros cenários que poderiam ser avaliados diretamente com a mesma abordagem. No nível de rede, por exemplo, seria possível testar um *flood* de requisições, com múltiplas conexões abertas em curto intervalo, impactando métricas como `n_communication` e volume de tráfego, ou ainda conexões persistentes de longa duração, alterando métricas de duração dos fluxos. No nível de aplicação, podem ser explorados casos como envio de queries excessivamente grandes, repetição contínua de chamadas ao mesmo endpoint em alta frequência ou sequências de requisições inválidas que geram erros sistemáticos. Esses comportamentos incluem tanto ataques quanto falhas operacionais, como clientes mal implementados gerando loops de requisição, e podem ser incorporados como novos cenários sem modificações no pipeline.

4.4.1 Limitações

A abordagem apresenta limitações associadas às escolhas metodológicas adotadas na modelagem do comportamento benigno e na definição das features monitoradas. A capacidade de detecção depende diretamente da linha de base construída, da janela temporal utilizada e das métricas extraídas nos níveis de rede e aplicação. Como consequência, desvios que não produzam alterações relevantes nessas variáveis podem não ser identificados com clareza. Da mesma forma, comportamentos benignos que se afastem do padrão observado durante a fase de referência podem aumentar a ocorrência de sinalizações indevidas.

Por exemplo, um ataque de enumeração realizado de forma muito espaçada no tempo, com poucas requisições distribuídas ao longo de um intervalo maior, pode não produzir alterações suficientemente fortes nas métricas monitoradas para ser claramente distinguido do comportamento benigno. Por outro lado, atividades benignas incomuns, como sessões de teste, manutenção ou diagnóstico com muitas requisições concentradas em pouco tempo, podem produzir falsos positivos. Além disso, a execução dos experimentos em um ambiente isolado, como a Raspberry Pi dedicada ao sistema, reduz fontes externas de variação no tráfego observado. Em ambientes compartilhados, nos quais coexistem comunicações geradas por múltiplos processos e serviços, a separação entre comportamento normal e anômalo tende a se tornar mais difícil.

Como o trabalho envolveu a construção de todo o pipeline experimental, desde o desenvolvimento da aplicação e dos dispositivos simulados até a definição das estratégias de

monitoramento, coleta de dados, agregação temporal, construção do dataset, treinamento dos modelos e execução da inferência em tempo real, a profundidade dedicada a cada uma dessas etapas foi necessariamente limitada. Nesse sentido, a solução deve ser entendida como uma prova de conceito funcional, construída de forma holística para integrar todas as peças do problema, mas ainda com muito espaço para refinamentos específicos em diferentes frentes. Aspectos como a definição de features e estratégias de janelamento em nível de rede, o ajuste mais fino dos modelos de aprendizado e o enrobustecimento da própria aplicação podem ser aprofundados por pessoas dedicadas a essas áreas.

4.4.2 Trabalhos Futuros

Um possível encaminhamento para trabalhos futuros consiste na realização de uma avaliação mais sistemática dos modelos, com base em um conjunto rotulado de janelas benignas e anômalas. Isso permitiria medir de forma mais objetiva o desempenho do sistema e comparar seus resultados em diferentes cenários de detecção. Além disso, seria relevante ampliar o conjunto de anomalias consideradas, incorporando comportamentos mais variados e sutis, de modo a avaliar com maior abrangência a capacidade de generalização da abordagem frente a diferentes padrões de desvio.

Outra direção promissora está na evolução da própria arquitetura de detecção. Nesse sentido, podem ser investigadas estratégias de fusão entre os níveis de rede e aplicação, buscando combinar suas evidências de forma mais integrada. Também seria relevante analisar a sensibilidade do sistema a parâmetros como tamanho da janela temporal, limiar de decisão e conjunto de features adotado. Por fim, a comparação com outros modelos de detecção pode contribuir para avaliar em que medida a abordagem baseada em Autoencoders representa a alternativa mais adequada para o problema.

5 Conclusão

Este trabalho apresentou uma prova de conceito para detecção de anomalias em um ambiente experimental que simula um sistema distribuído, atingindo os objetivos propostos de modelar o comportamento benigno, definir métricas representativas e utilizar modelos Autoencoder para identificar desvios ao longo do tempo. A abordagem adotada demonstrou ser capaz de capturar padrões de funcionamento do sistema e detectar comportamentos anômalos em diferentes cenários, validando a viabilidade da estratégia baseada em aprendizado não supervisionado e análise por janelas temporais.

O desenvolvimento da solução envolveu a integração de diferentes domínios de conhecimento, incluindo redes de computadores, segurança de sistemas, aprendizado de máquina e implementação de software. Essa combinação foi necessária para construir o ambiente experimental e analisar o comportamento do sistema, e proporcionou uma experiência enriquecedora ao exigir o contato com diferentes áreas e a articulação desses conhecimentos em um único contexto, contribuindo para uma compreensão mais holística e integrada de sistemas computacionais.

Dessa forma, o trabalho cumpre seu propósito como prova de conceito, mostrando na prática que a modelagem do comportamento e o monitoramento em múltiplos níveis permitem identificar diferentes tipos de desvio no sistema. Os resultados obtidos evidenciam que as perspectivas de rede e aplicação são complementares e, quando utilizadas em conjunto, fornecem uma visão mais completa do comportamento observado. Com isso, o trabalho consolida uma base sólida para análise comportamental em sistemas distribuídos dentro do escopo proposto.

Referências

CWE-1336. *CWE-1336: Improper Neutralization of Special Elements Used in a Template Engine*. 2026. <<https://cwe.mitre.org/data/definitions/1336.html>>. Acessado em março de 2026. Citado na página 30.

CWE-94. *Improper Control of Generation of Code ('Code Injection')*. 2026. <<https://cwe.mitre.org/data/definitions/94.html>>. Acessado em março de 2026. Citado na página 30.

Keras Documentation. *Autoencoder Example*. 2026. <<https://keras.io/examples/vision/autoencoder/>>. Acessado em março de 2026. Citado na página 15.

National Institute of Standards and Technology (NIST). *Guide to Intrusion Detection and Prevention Systems (IDPS)*. [S.l.], 2007. Disponível em: <<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf>>. Citado 2 vezes nas páginas 14 e 15.

National Institute of Standards and Technology (NIST). *NIST Computer Security Resource Center Glossary*. 2026. <<https://csrc.nist.gov/glossary>>. Acessado em março de 2026. Citado na página 14.

NFStream Project. *NFStream: Flexible Network Data Analysis Framework*. 2026. <<https://github.com/nfstream/nfstream>>. Acessado em março de 2026. Citado na página 23.