RAVEN: Detecção e Classificação Precoce de Atores Maliciosos em uma Rede Acadêmica

Willen B. Coelho[1,2], Vitor F. Zanotelli[2], Giovanni Comarela[2], Rodolfo S. Villaça[2]

[1] Instituto Federal do Espírito Santo (Ifes)
Campus Cachoeiro do Itapemirim

[2]Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Espírito Santo (Ufes)

willen@ifes.edu.br
vitor.zanotelli@edu.ufes.br
{giovanni.comarela,rodolfo.villaca}@ufes.br

Resumo. *A varredura de portas é uma importante técnica de coleta de informações sensíveis, por isso, destacamos a necessidade de sistemas aprimorados de segurança e ressaltamos que a varredura, considerada uma anomalia, deve ser identificada e suprimida precocemente, especialmente diante do número significativo de incidentes reportados. Em resposta a esse desafio, propõe-se um sistema inteligente e automatizado que analisa fluxos de rede para detectar e classificar varreduras no menor tempo possível. As contribuições incluem a implementação e avaliação do sistema, a demonstração da melhoria do desempenho com a ampliação das features e a disponibilização de conjuntos de dados para a comunidade acadêmica.*

Abstract. *Port scanning is an important technique for gathering sensitive information. We underscore the need for enhanced security systems, as port scanning, though an unusual activity, should be identified and suppressed early, especially given the number of reported incidents. In response to this challenge, we propose an intelligent and automated system that analyzes network traffic to detect and classify port scans in near real-time. Our contributions include the implementation and evaluation of the online system, demonstrating how the inclusion of more information improves performance, and realeased datasets for the academic community.*

1. Introdução

Num esforço para atingir alvos específicos de ataque, criminosos cibernéticos buscam explorar vulnerabilidades de segurança em sistemas computacionais. Conforme [Strom et al. 2018], a varredura de rede visando a descoberta de vulnerabilidades é um estágio inicial importante em qualquer ataque cibernético, e o CERT.br [CERT.br 2023] divulgou recentemente que, apenas em 2023, foram reportados um total de 581.038 incidentes. Desse total, 74,08% representam incidentes de varredura.

A varredura de rede é uma importante técnica de coleta de informações sensíveis. Portanto, mitigar essa fase de reconhecimento dos sistemas é uma estratégia eficaz de defesa da rede. De acordo com [Cabaj et al. 2018], geralmente, a varredura precede ataques reais e, dada a persistência das ameaças atuais, a implementação de sistemas aprimorados

de segurança se faznecessária, visto que outras tecnologias, tais como antivírus e *firewalls* podem não ser suficientes. Griffioen [Griffioen and Doerr 2020] destaca que, para não serem detectados, os atacantes podem retardar a varredura ou distribuí-la por múltiplos *hosts* de origem, dificultando, assim, a utilização de sistemas de monitoramento.

Este cenário exige um sistema inteligente e automatizado para monitorar o tráfego, preferencialmente em tempo real, detectar e classificar atividades maliciosas do tipo varredura, permitindo que os administradores de rede possam mitigá-las precocemente. Nesse contexto surge uma importante questão de pesquisa: "Como detectar e classificar os diferentes ataques de varredura, com bom desempenho, e no menor tempo possível?".

Visando responder esse questionamento, este artigo apresenta o RAVEN (*Rapid Analysis and Verification of EmergiNg threats*), um sistema capaz de analisar as informações de fluxo para detectar e classificar ataques de varredura, no menor tempo possível. Além disso, é requisito do RAVEN que a operação ocorra sem a necessidade de realização de inspeção de pacotes, respeitando a privacidade dos usuários da rede. Este artigo possui, como hipótese de desenvolvimento, a utilização de informações de fluxo como recurso para treinamento de modelos de Aprendizado de Máquina (ML). Em particular, ao final deste artigo pretende-se responder às seguintes questões de pesquisa:

- Pode-se detectar e classificar, precocemente, diferentes tipos de varredura, com base apenas nas informações de fluxo usando algoritmos de ML?
- Quais são as características dos fluxos que ajudam o treinamento e aumentam o desempenho dos modelos de ML para detecção e classificação de varreduras?
- É possível generalizar um modelo de ML para maximizar as características específicas e particulares de cada varredura e, ao mesmo tempo, evitar o *overfitting*?

As contribuições deste trabalho podem ser sintetizadas, da seguinte forma: i) implementação e avaliação do RAVEN na rede *eduroam* do Ifes; ii) demonstração, por meio de experimentos, de que a ampliação das características básicas dos fluxos, fornecidas pelos sistemas de monitoramento, e posterior seleção com redução de dimensionalidade, melhora consideravelmente o desempenho dos modelos treinados; iii) geração de 2 conjuntos de dados[1], com parâmetros distintos, simulando diferentes configurações de ataques, e com informações de tráfego real, composto por fluxos normais da rede *eduroam*.

O restante deste artigo está organizado da seguinte forma: Na Seção 2, são apresentados trabalhos relacionados. A Seção 3 descreve a metodologia para geração do *dataset*, enquanto, a Seção 4 apresenta as fases de pré-processamento, seleção de características e os experimentos realizados. A Seção 5 descreve a implantação do RAVEN na rede do Ifes. E, por final, a Seção 6 resume as conclusões e discute trabalhos futuros.

2. Trabalhos Relacionados

A varredura é uma técnica frequentemente adotada, tornando-se um ponto central para pesquisas em detecção de atores maliciosos. Segundo [Bou-Harb et al. 2013], a varredura, é a tarefa de escanear e analisar redes ou serviços, em busca de vulnerabilidades ou fragilidades dos ativos de TI. E isso deve-se ao fato de a varredura ser normalmente a fase primária de uma tentativa de intrusão que permite a um atacante localizar,

[1] `https://github.com/spawnzao/sbrc2024`

atacar e subsequentemente explorar remotamente sistemas vulneráveis. De acordo com [Bhuyan et al. 2011], essencialmente, uma varredura consiste em enviar uma mensagem para cada porta e aguardar uma resposta. O tipo de resposta recebida pode levar a novas varreduras com o intuito de investigar as fraquezas, para lançar ataques futuros.

Além disso, soluções de detecção de ataques com ML, são frequentemente investigados. [Mishra et al. 2019] fornece uma revisão dos métodos de ML para IDSs, associando diferentes tipos de ataques aos recursos que podem ser usados para detectá-los. [Devan and Khare 2020] concentram-se na detecção de ataques, incluindo varreduras, utilizando uma abordagem de ML que combina o XGBoost e *Deep Neural Network* (DNN) e se baseia no conjunto de dados NSL-KDD. Por outro lado, Liu *et al.* [Liu et al. 2021] abordam diferentes conjuntos de dados NSL-KDD, UNSW-NB15 e CICIDS2017, que incluem varreduras, empregando uma combinação do algoritmo LightGBM com a técnica de geração sintética ADASYN. Satheesh *et al.* [Satheesh et al. 2020] utilizam o protocolo Openflow, emulação no Mininet e exploram uma variedade de ataques por meio de modelos tradicionais de classificação. Mais recentemente, [Araujo et al. 2023] apresentam o ANTE, que se destaca por selecionar autonomamente o pipeline de ML mais apropriado para cada *botnet*, aprimorando a classificação com *features* de fluxo, antes que o ataque efetivamente aconteça. São utilizados 4 conjuntos de dados: ISOT HTTP Botnet, CTU-13, CICDDoS2019, and BoT-IoT.

No cenário de trabalhos com redes reais, [Do and Gadepally 2020] usam as entropias da rede com DNN para a detecção de anomalias, extraindo características dos fluxos de rede, enquanto [Camacho et al. 2019] propõem um método de detecção que utiliza a técnica *Multivariate Big Data Analysis* (MBDA) associado a informações de rede e dados *NetFlow* como *features*. O estudo abrange também uma ampla gama de ataques, incluindo varredura.

Concluindo, a partir da análise dos trabalhos relacionados, muitos estudos se concentram em abordagens abrangentes para variados tipos de ataques, enquanto técnicas específicas de varreduras permanecem menos explorados. Além disso, a investigação de ataques em redes reais, adicionada a análise em tempo real, representam um desafio. Nesse sentido, o RAVEN foi proposto para solucionar esses problemas, destacando-se pela análise criteriosa das *features*, a geração de dois conjuntos de dados com tráfego real, a busca pela generalização dos modelos e sua aplicação *online*, características que tornam o RAVEN uma abordagem inovadora e promissora no campo da detecção de varreduras ativas e furtivas de rede.

3. Geração dos Conjuntos de Dados

Nesse trabalho são criados dois conjuntos de dados (*dataset*), gerados a partir da mistura de tráfego malicioso, contendo diferentes tipos de ataques de varredura, com o tráfego real dos usuários da rede sem-fio acadêmica (*eduroam*) do Ifes, *campus* Cachoeiro de Itapemirim/ES[2], que possuia mais de 3200 alunos matriculados em 2023. Estes conjuntos de dados foram gerados em datas diferentes visando capturar propriedades distintas do tráfego normal da instituição e permitir uma maior generalização dos modelos treinados. O primeiro conjunto de dados contém informações de fluxo de aproximadamente 560

[2]Importante destacar que esta pesquisa foi realizada com autorização da direção do *campus* durante 2 dias do mês de novembro/2023.

dispositivos de rede, enquanto o segundo contém dados de fluxo de aproximadamente 440 dispositivos, tais como computadores, *notebooks*, celulares, *tablets*, *smartwatches* e equipamentos IoT.

É importante destacar a necessidade de gerar dados variados e evitar o *overfitting*. Visando maximizar os tipos e características das varreduras, e generalizar o modelo de detecção e classificação, a geração do *dataset* focou, principalmente, em variar os padrões particulares e específicos de cada tipo de varredura. Os destinos dos ataques também foram modificados e variados entre os dois conjuntos de dados usados para treino e teste. Além do tráfego normal, a atividade maliciosa é composta por 8 dispositivos distintos, que realizaram 8 diferentes tipos de varredura, usando técnicas variadas, que serão melhor detalhadas na Seção 3.2.

3.1. Metodologia

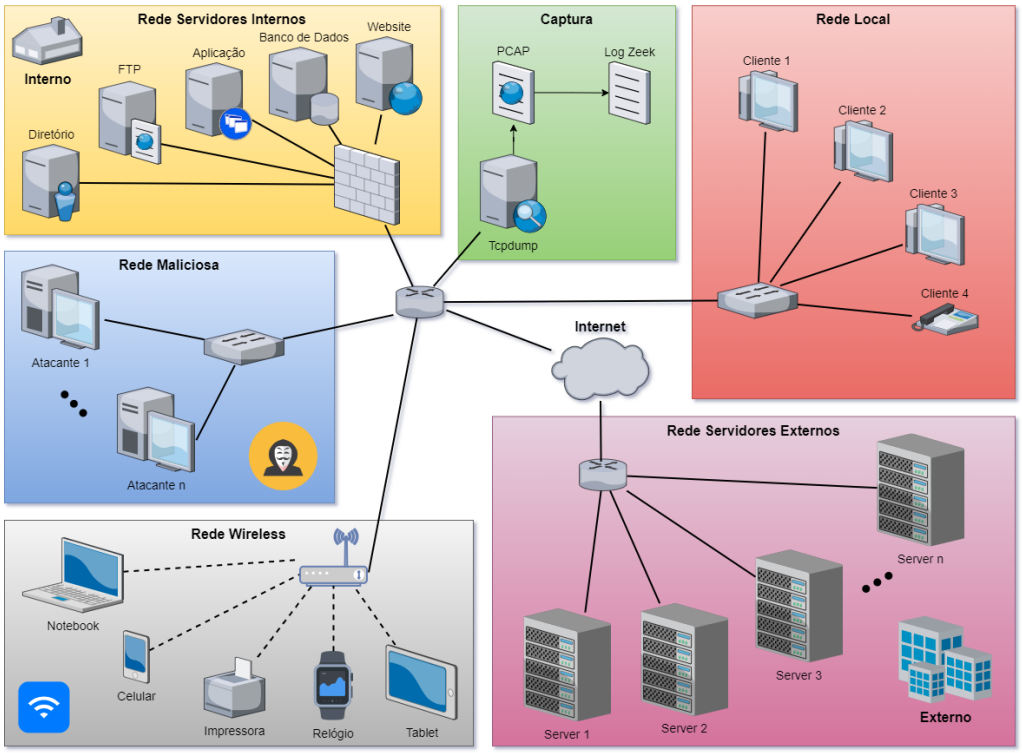


Figura 1. Infraestrutura da rede da instituição acadêmica usada para a geração dos conjuntos de dados de treino e teste.

Conforme mostrado na Figura 1, a infraestrutura de rede utilizada para a geração dos conjuntos de dados pode ser dividida em quatro subgrupos: i) "Rede Interna", formada pelo conjunto `Rede Local`, `Rede Wireless` (*eduroam*) e a `Rede Servidores Internos`; ii) "Rede Externa", formada por servidores em outro campus da instituição, acessíveis publicamente pela Internet, denominada `Rede Servidores Externos`; iii) Rede `Captura`, formada por uma máquina com acesso de espelhamento de pacotes com as demais redes, permitindo a captura de todo o tráfego; e iv) `Rede Maliciosa`, composta pelos 8 dispositivos atacantes que gerarão o tráfego malicioso de varredura nas redes interna e externa.

A ferramenta *tcpdump* foi utilizada para coleta de pacotes na rede *eduroam* a partir da `Rede Captura`. Para extrair informações de fluxo a partir dos pacotes capturados, a ferramenta *Zeek* foi utilizada, produzindo as informações básicas de fluxo em arquivos de *log*. Adicionalmente ao *Zeek*, características[3] adicionais foram incluídas ao conjunto de dados por meio da extensão *FlowMeter* do Zeek [Habibi Lashkari et al. 2017], havendo assim. inicialmente uma expansão do conjunto de características. Mais especificamente, neste trabalho foram gerados dois arquivos de *log* de interesse: i) *conn.log* (somente *Zeek*), contendo informações de conexão; e ii) *flowmeter.log* (*FlowMeter*), contendo informações de tráfego referentes às camadas de rede e transporte. Exemplificando, o *conn.log* fornece os dados relativos ao tráfego entre dois *hosts*, como endereços IP, duração, quantidade de *Bytes* trafegados, estado do fluxo, quantidade de pacotes e informações de tunelamento. Já o *flowmeter.log* fornece metadados de fluxo que incluem a proporção do número de pacotes de retorno e encaminhamento, quantidade de *Bytes* nos cabeçalhos, tamanho da carga útil, sinalizadores (*flags*) de um fluxo TCP, tempo entre chegadas de pacotes consecutivos do mesmo fluxo, quantidade de *Bytes* de carga e duração do fluxo.

Cada atacante gerou um tipo específico de varredura, permitindo a rotulagem automática com base no endereço IP de origem do fluxo malicioso. É importante considerar que é sabido pelos autores que esta abordagem pode gerar tendências, vieses e padrões específicos de tráfego e, a fim de tornar o modelo o mais genérico possível, e aplicável em outros ambientes de rede, várias características do fluxo (*features*) foram removidas do conjunto de dados, mais especificamente: todas as características relacionadas com a identificação do atacante (MAC, IP, portas), características repetidas e todas as *features* temporais relacionadas ao momento de geração do ataque, e atributos de vazão, velocidade e largura de banda da rede. Essa abordagem foi utilizada para evitar que o modelo aprenda padrões inerentes do ambiente de geração dos ataques.

Portanto, no *dataset* de treino utilizou-se como alvo a "Rede Interna" e para o *dataset* de teste utilizou-se como alvo a "Rede Externa". Além disso, a frequência dos ataques, quantidade de atacantes executando em paralelo, intervalo de portas varridas, endereços IP de destino e a duração das varreduras, também variou em ambos os *datasets*. Contudo, vale destacar, que informações sensíveis foram anonimizadas, evitando assim a exposição desnecessária a possíveis vazamentos de informação, conforme preconizam as práticas, princípios e exigências da Lei Geral de Proteção de Dados (LGPD).

3.2. Tipos de Ataques de Varredura

Para a realização das varreduras utilizou-se o software *nmap*, que oferece diversas opções e técnicas de geração. Em ambos os conjuntos de dados foram realizadas varreduras verticais (múltiplas portas de um único *host* específico) e horizontais (uma única porta específica de múltiplos *hosts*), isto é, múltiplas portas e múltiplos destinos foram definidos.

É importante destacar que na geração dos conjuntos de dados foram utilizados 8 tipos diferentes de varredura[4] (*FIN, XMAS, NULL, SYN, CONNECT, ACK, UDP, Mainmon*), todos devidamente rotulados. No primeiro conjunto de dados, selecionado para treinar o modelo, utilizaram-se as seguintes configurações de geração: i) varredura nas

[3]Os termos características, atributos e *features* serão utilizados no mesmo sentido semântico.

[4]`https://nmap.org/book/man-port-scanning-techniques.html`

200 portas mais comumente utilizadas; ii) paralelismo dos atacantes com um máximo de 300 processos simultâneos; iii) controle de temporização T5 (*Insane*), visando intensificar os ataques durante o treinamento e iv) alvo da varredura "Rede Interna". O segundo conjunto de dados foi selecionado para teste, isto é, para avaliar o modelo criado. Por se tratar de uma rede externa, a fim de escapar da detecção por possíveis sistemas de monitoramento, reduziu-se a quantidade de portas varridas e optou-se por uma configuração de temporização mais lenta. Sendo assim, as configurações utilizadas são: i) as 50 portas; ii) sem paralelismo; iii) controle de temporização T1 (*Sneaky*) e T2 (*Polite*); iv) alvo da varredura: "Rede Externa". As descrições de cada controle de temporização são mostradas na Tabela 1. Em geral, quanto maior o nível, maior será o consumo de largura de banda e a sobrecarga no *host* atacante.

Tabela 1. Caracterização dos diferentes controles de temporização usados na geração dos ataques de varredura através da ferramenta *nmap*.

Temporização	Descrição
T1 (Sneaky)	O intervalo de envio de cada varredura é de 15 s.
T2 (Polite)	O intervalo de envio de cada varredura é de 0,4 s.
T3 (Normal)	Varredura mais rápida possível, modelo padrão.
T5 (Insane)	Semelhante ao T3, mas com o tempo máximo de atraso de 5 ms.

4. Geração dos Modelos de Classificação dos Ataques de Varredura

Esta seção descreve a metodologia usada nas fases de pré-processamento, estratégias de seleção de características e o desenvolvimento do modelo de treinamento e teste, juntamente com a análise de desempenho dos modelos e a apresentação dos resultados obtidos.

4.1. Pré-Processamento

Após a coleta dos pacotes e geração das informações de fluxo, os *logs* do *Zeek* + *FlowMeter* foram transformados, no qual cada coluna representa uma *feature* e cada linha representa um fluxo de rede. Os valores ausentes no conjunto de dados foram zerados. Uma nova coluna (rótulo, ou *target*), denominada *scan_type*, foi adicionada, rotulando cada um dos 8 diferentes tipos de varredura presentes no conjunto de dados. Foi adicionado também um novo rótulo, chamado Y, usado apenas para o treinamento dos modelos de classificação binária (i.e., $Y = 0$ para fluxos normais e $Y = 1$ para varredura).

O conjunto de dados derivado do *conn.log* (somente *Zeek*) possui originalmente 18 *features* básicas e o conjunto de dados expandido, *flowmeter.log*, possui originalmente 80 *features*. Após a etapa de união entre os dois conjuntos de dados, no qual é utilizado um identificador exclusivo para relacionar cada fluxo nos diferentes arquivos, é formado um novo conjunto de dados com 98 *features*. Após a conversão das variáveis categóricas, através da técnica *one-hot encoding*, o número de atributos aumenta para 144.

Além disso, o conjunto de dados de treino possui um total de 654.947 fluxos, sendo eles: i) 571.239 (87,2%) de tráfego normal e ii) 83.708 (12,8%) de tráfego de varredura, enquanto o conjunto de dados de teste possui um total de 590.136 fluxos, sendo eles: i) 573.418 (97,2%) de tráfego normal e ii) 16.718 (2,8%) de tráfego de varredura. A Tabela 2 fornece as estatísticas detalhadas destes quantitativos nos conjuntos de dados de treinamento e teste. Vale ressaltar que, propositalmente, com o intuito de simular uma

Tabela 2. Distribuição estatística dos fluxos normal e de varredura nos conjuntos de dados de treino e teste.

Dataset	Normal	FIN	XMAS	NULL	SYN	CONNECT	ACK	UDP	Mainmon
Treino	571.239 (87,2%)	11.522 (1,8%)	10.904 (1,7%)	10.808 (1,7%)	10.898 (1,7%)	11.553 (1,8%)	9.989 (1,5%)	8.354 (1,3%)	9.680 (1,5%)
Teste	573.418 (97,2%)	1.759 (0,3%)	1.746 (0,3%)	1.754 (0,3%)	1.896 (0,3%)	3.815 (0,6%)	1.755 (0,3%)	2.247 (0,4%)	1.746 (0,3%)

varredura furtiva mais realista e evitar o *overfitting* do modelo, a proporção entre os fluxos normais e maliciosos no conjunto de dados de teste é diferente da proporção presente no conjunto de dados de treinamento.

4.2. Seleção de Características

Afim de diminuir o *overfitting* (ou sobreajuste) de modelos de aprendizado buscou-se a redução da dimensionalidade do conjunto de dados, eliminando atributos menos significativos, ou irrelevantes, para evitar o ajuste excessivo do modelo e melhorar o desempenho da classificação. Portanto, a primeira fase da seleção de características foi a remoção de colunas desnecessárias, utilizando-se como base o conjunto de dados de treino, completo, formado por rótulos da junção *Zeek + FlowMeter*.Conforme já descrito anteriormente, as colunas diretamente relacionadas com a infraestrutura de rede e com método de geração das varreduras também foram removidas. A exclusão dessas *features* temporais é crucial, pois fluxos de rede provenientes de diferentes origens (externas ao alvo da varredura) poderiam introduzir variações temporais que poderiam facilitar o aprendizado com padrões viciados, e reduzir a capacidade do modelo de aprender padrões consistentes. Ao remover essas influências temporais específicas, buscamos garantir uma maior generalização e aplicabilidade do modelo em situações diversas ao treinamento realizado.

Na segunda fase da seleção de características, os atributos correlacionados linearmente entre si foram removidos. Empregou-se o Coeficiente de Correlação de Pearson para calcular a relação entre os pares de atributos, resultando na remoção de um dos elementos, de cada par, com coeficientes de correlação próximos a 1.

Na terceira fase, a função *get_score* da biblioteca XGBoost foi utilizada para estimar a importância das *features* ainda restantes no conjunto de dados, atribuindo uma pontuação para cada uma delas com base na sua importância para a geração de um modelo de classificação, como pode ser visto na Figura 2. Basicamente, o parâmetro peso (*weight*) conta quantas vezes o atributo é escolhido para fins de divisão nas diversas árvores de decisão do modelo gerado. Já o parâmetro ganho (*gain*) representa o ganho médio de informação nos modelos em que a *feature* é utilizada. Por fim, a cobertura (*cover*) é a cobertura média dos modelos em que uma *feature* específica é utilizada. A partir da figura, foi possível observar a importância da extensão das caraterísticas dos fluxos, obtida com o uso da combinação *Zeek + FlowMeter*, pois, segundo essas métricas, a maioria das caraterísticas relevantes para o modelo são provenientes dessa extensão. A figura representa apenas uma parte das *features* selecionadas em um total de 91/144 características.

4.3. Treinamento e Teste

Os modelos de classificação dos ataques de varredura desenvolvidos neste trabalho foram gerados por meio de algoritmos de aprendizado de máquina supervisionados, a partir

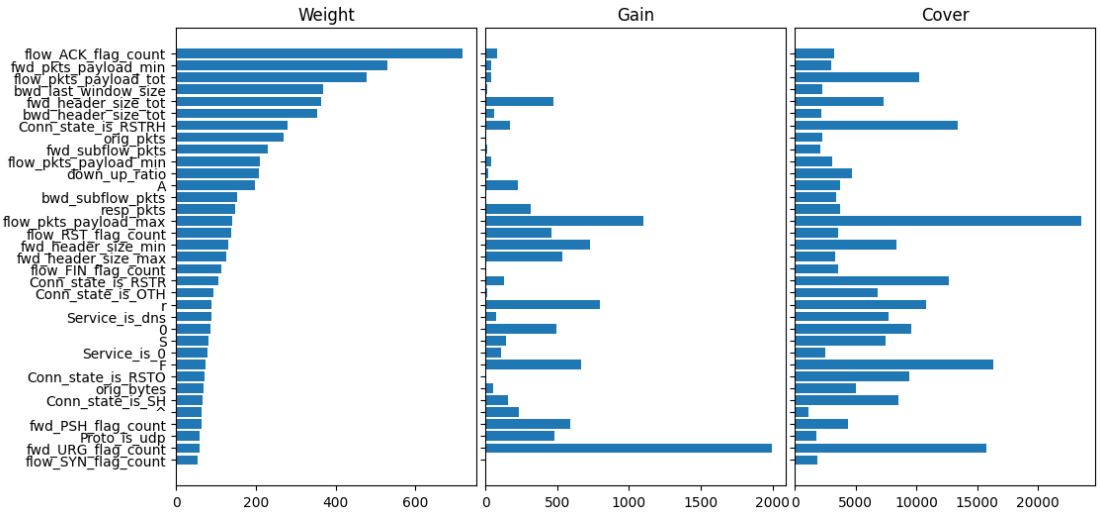


Figura 2. Importância das *features*

da seleção dos melhores atributos encontrados no processo de seleção descrito na seção anterior. É importante relembrar que os conjuntos de dados de treino e teste, gerados e utilizados neste artigo, são totalmente diferentes entre si, obtidos em dias diferentes da semana, com características de tráfego normal bastante heterogêneos. Além disso, o tráfego malicioso também sofreu diversas modificações em suas características para simular um comportamento diversos dos atacantes envolvidos no treinamento e evitar vazamento de dados entre as *features* dos fluxos normal e malicioso.

Para realizar o treinamento dos modelos de classificação dos ataques de varredura foram utilizados os seguintes algoritmos de aprendizado de máquina: *XGBoost, Random Forest, Adaptive Boosting* e *Bagging* pois foram os algoritmos que obtiveram os melhores resultados em um conjunto de testes preliminares (não apresentados aqui devido a restrições de espaço), juntamente com o algoritmos *Naive Bayes*, usado apenas como referência de desempenho (*baseline*) dos modelos.

Para avaliar melhor a previsibilidade dos modelos e a eficiência na detecção de diferentes tipos de varredura de rede, três experimentos foram realizados. No primeiro experimento (Experimento I) foram usados apenas os atributos básicos de um fluxo, produzidos somente pela ferramenta *Zeek*, na classificação binária. O segundo experimento (Experimento II) visa avaliar a importância da expansão das *features* originais, usando características adicionais, geradas pela extensão *Zeek + FlowMeter* na classificação binária dos fluxos. Por fim, o terceiro e último experimento (Experimento III) também foi realizado com o *dataset* expandido e visando a geração de um modelo de classificação multiclasse (9 classes), capaz de identificar cada um dos 8 tipos de ataques de varredura. O treinamento foi realizado com uso da linguagem de programação *Python* e das bibliotecas *Pandas, XGBoost* e *Scikit-learn*, dentre outras.

O desempenho dos modelos de classificação binária foi representado por meio da geração da Matriz de Confusão de cada modelo, contendo os Verdadeiros Positivos (TP), Falsos Positivos (FP), Falsos Negativos (FN) e Verdadeiros Negativos (TN). As seguintes métricas foram utilizadas na avaliação de desempenho dos modelos: Acurácia, Precisão, Revocação e *F1-score*. Conforme já detalhado anteriormente, como foram gerados 2

(dois) *datasets*, um para treino e outro para teste, nenhuma técnica de amostragem para separação dos conjuntos de dados foi necessária. Assim, não houve necessidade de se avaliar a independência entre os conjuntos de treinamento e teste. Nos três experimentos utilizou-se os mesmos conjuntos de dados e os mesmos algoritmos de aprendizado, garantindo assim a justiça na avaliação de desempenho dos modelos treinados.

4.3.1. Experimento I: Classificação Binária (Somente *Zeek*)

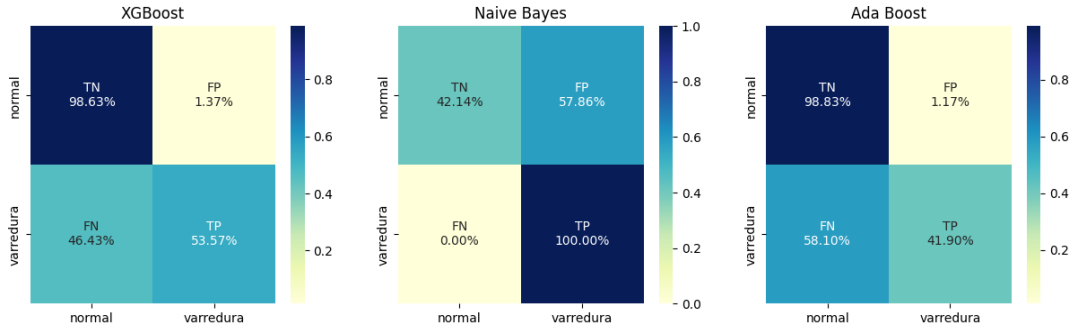


Figura 3. Matriz de confusão dos modelos binários, no Experimento I, usando apenas as *features* originais, disponibilizadas pela ferramenta *Zeek*.

O primeiro experimento teve como objetivo classificar corretamente os tráfegos normal e de varredura, usando apenas as características básicas de fluxo geradas por meio da ferramenta *Zeek*. Na Figura 3 são apresentadas as matrizes de confusão para os algoritmos *XGBoost, Naive Bayes* e *Adaptative Boost*. Vale ressaltar, que as matrizes de confusão dos algoritmos *Random Forest* e *Bagging* não foram apresentados por possuírem resultados muito similares aos do *XGBoost*, e devido às restrições de espaço no artigo.

Na Tabela 3 são apresentados as métricas de desempenho para todos os modelos de classificação binária gerados neste experimento. Os algoritmos *Random Forest, Bagging* e *XGBoost* foram os que apresentaram a maior acurácia. Entretanto, também é essencial avaliar os valores de Precisão, Revocação e *F1-Score* para identificar os pontos fortes e fracos de cada modelo e compreender totalmente a qualidade dos resultados obtidos.

Tabela 3. Resultados da Avaliação de Desempenho dos modelos binários produzidos no Experimento I. Resultados contém apenas os atributos básicos de fluxo (*Zeek*).

Modelo	Acurácia	Precisão	Revocação	F1-score	Tempo	
					Média	Desvio
XGBoost	97,3%	53,3%	53,6%	53,4%	2,73 s	0,06 s
Bagging	97,3%	53,2%	53,6%	53,4%	10,82 s	0,29 s
Random Forest	97,2%	53,2%	53,4%	53,4%	6,70 s	0,06 s
Ada Boost	97,2%	51,0%	41,9%	46,0%	8,10 s	0,04 s
Naive Bayes	43,8%	4,8%	100%	9,2%	0,85 s	0,02 s

Vê-se claramente que existem diferenças mínimas entre os valores de Acurácia, Precisão, Revocação e *F1-Score* dos modelos *XGBoost, Random Forest* e *Bagging*. Também foi realizada uma avaliação da métrica de tempo de predição desses modelos,

resultado também apresentado na Tabela 3. Para poder comparar quantitativamente o tempo médio de predição dos modelos, consideramos o tempo decorrido na função de predição de todo o conjunto de teste, coletado através da a função *perf_counter* da biblioteca *time* do *Python*. Essa medição foi realizada 100 (cem) vezes. Com base nestes resultados, pode-se afirmar que o *XGBoost* atingiu o melhor desempenho e menor tempo de execução neste experimento e será usado no Experimento II, descrito a seguir.

4.3.2. Experimento II: Classificação Binária (*Zeek + FlowMeter*)

No Experimento II, foi avaliada a importância da ampliação do número de atributos de fluxo, gerados pela união das ferramentas *Zeek + FlowMeter*, na geração de um modelo de classificação binária. Dessa forma, comparou-se o resultado do treinamento obtido por meio das *features* exclusivas da ferramenta *Zeek* (i.e., features derivadas somente do arquivo *conn.log*), com resultado do treinamento obtido por meio das *features* expandidas (i.e., derivadas da junção do arquivo *conn.log* e do *flowmeter.log*). Para comparar os resultados de ambos modelos utilizou-se somente o *XGBoost*.

Tabela 4. Comparativo estatístico entre o treinamento com diferentes conjuntos de atributos, conforme descrito no Experimento II. `Somente Conn` é Zeek, enquanto `Conn + Flowmeter` é *Zeek + FlowMeter*.

Atributos	Acurácia	Precisão	Revocação	F1-Score
`Somente Conn`	97,4%	53,3%	53,6%	53,4%
`Conn + FlowMeter`	98,8%	81,6%	71,9%	76,5%

Na Figura 4, são apresentadas as matrizes de confusão obtidas com o treinamento de ambos os conjuntos de atributos, e é possível observar uma diferença significativa na quantidade de amostras classificadas como Normal e Varredura, bem como, uma diferença importante nas quantidades de amostras nos parâmetros FP e FN, da matriz de confusão. Conforme esperado, os modelos gerados com o XGBoost obtiveram desempenho superior quando treinados e testados com o conjunto de dados expandido. A Tabela 4 resume os resultados obtidos com este experimento.

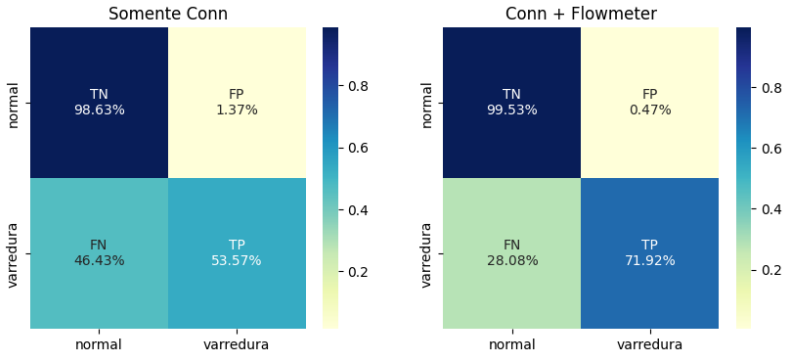


Figura 4. Comparativo entre as matrizes de confusão dos modelos gerados no Experimento II a partir de diferentes conjuntos de características: *Zeek* (Somente *conn.log*) e *Zeek + FlowMeter* (*conn.log + flowmeter.log*).

A partir dos resultados apresentados na Tabela 4, a expansão das *features* parece ser a melhor alternativa para a modelagem do problema proposto, já que os resul-

Tabela 5. Estatísticas de desempenho dos modelos multi-classe, treinados com diferentes algoritmos, conforme descrito no Experimento III.

Modelo	Acurácia	Ponderado			Tempo	
		Precisão	Revocação	F1-score	Média	Desvio
XGBoost	98,8%	98.7%	98.8%	98.6%	6,14 s	0,21 s
Bagging	98.8%	98,7%	98,8%	98,6%	18,63 s	0,04 s
Random Forest	98,8%	98,7%	98,8%	98,6%	10,30 s	0,05 s
Ada Boost	95,4%	95,3%	95,4%	95,3%	17,92 s	0,11 s
Naive Bayes	44,2%	97,2%	44,2%	59,9%	5,17 s	0,07 s

tados demonstraram uma melhora relevante em relação aos resultados obtidos no Experimento I. Apesar do valor da acurácia demonstrar valores próximos, devido à diferença na proporção de fluxos normais e maliciosos nos *datasets*, as demais métricas de desempenho demonstram uma notável melhora no desempenho. Por isso, decidiu-se avaliar a capacidade de classificação multi-classe, separando o fluxo normal e os 8 tipos diferentes de ataques de varredura no próximo experimento.

4.3.3. Experimento III - Classificação Multi-Classe

No Experimento III passou-se a considerar a categoria da varredura, presente no rótulo *scan_type*, que possui nove classes. A classificação multi-classe permite uma avaliação mais detalhada, capaz de nos ajudar a responder ao seguinte questionamento: "os erros de classificação são pontuais (i.e., em poucas ou uma única categoria de varredura, por exemplo) ou gerais (i.e., em todas as categorias)?", além disso, nos ajudará investigar as causas das amostras classificadas incorretamente nos Experimentos I e II.

A partir dos resultados apresentados na Figura 5, pode-se concluir que a maioria das categorias de varredura (e.g., *FIN, XMAS, NULL, SYN* e *ACK*) e a categoria de tráfego normal possuem desempenho muito alto, com valores muito próximos a 100%. Pode-se notar, também, que as varreduras *CONNECT, UDP* e *MAILMON* possuem amostras preditas incorretamente (e.g., presença de FP ou FN), justificando a maioria das amostras classificadas incorretamente. São necessárias análises mais aprofundadas para investigar as causas e características destes ataques que levaram a esses resultados, porém estas análises estão fora do escopo deste artigo e serão exploradas em trabalho futuro.

A Tabela 5 apresenta um resumo dos resultados das métricas de desempenho dos modelos de classificação multi-classe. No entanto, as métricas de desempenho para esse experimento devem ser calculadas individualmente, por classe, e a avaliação ponderada calcula a média ponderada das métricas, levando-se em consideração a proporção do número de amostras de cada classe. Novamente, os algoritmos *Bagging, Random Forest* e *XGBoost* foram os que produziram os modelos com os melhores resultados. Na avaliação do tempo de predição dos modelos, o *XGBoost* obteve também o melhor desempenho.

Por fim, é importante esclarecer o seguinte: os Experimentos I e II, foram avaliados com base nas métricas de desempenho da classe positiva (varredura). No entanto, no Experimento III, que abrange um modelo multi-classe, a avaliação foi realizada considerando métricas de todas as classes de varredura, uma vez que há 8 (oito) classes positivas.

No geral, previsivelmente as métricas de desempenho do Experimento III, empre-

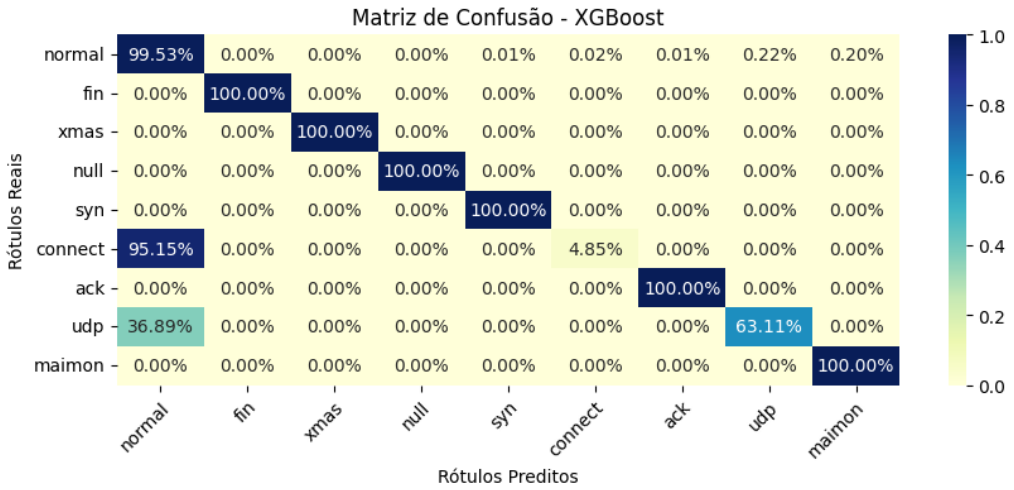


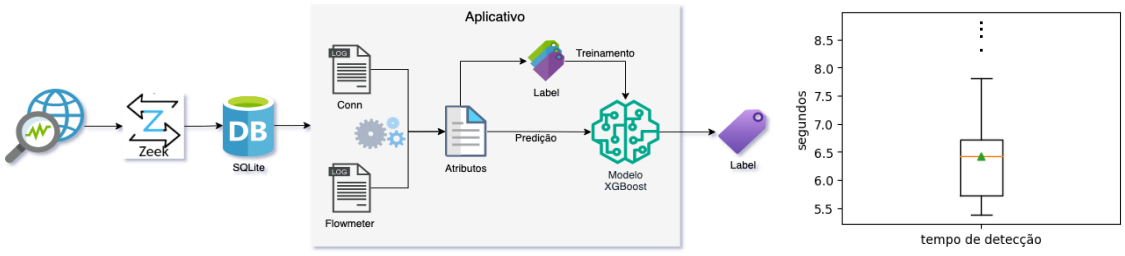
Figura 5. Matriz de confusão do algoritmo XGBoost no modelo multi-classe

Tabela 6. Resumo dos valores de *F1-score* obtidos nos diferentes Experimentos

Modelo	XGBoost	Bagging	Random Forest	Ada Boost	Naive Bayes
Experimento I	97,4%	97,4%	97,4%	97,1%	57,9%
Experimento II	98,8%	98,8%	98,8%	98,5%	59,7%
Experimento III	98,6%	98,6%	98,6%	95,3%	59,5%

gando o modelo multi-classe, são ligeiramente inferiores aos do Experimento II. Considerando apenas o modelo gerado pelo *XGBoost*, a métrica de F1-score ponderada possui valor de 98,6% na previsão multi-classe e 98,8% na previsão binária, conforme pode ser observado na Tabela 6. Além disso, na mesma tabela é possível confirmar a importância da expansão das *features* básicas que descrevem os fluxos, a partir dos piores resultados obtidos no Experimento I.

5. Implantação do RAVEN na Rede *eduroam* do Ifes



(a) (b)

Figura 6. a) Arquitetura do RAVEN. b) Diagrama de Caixa (*boxplot*) do tempo de resposta do RAVEN.

A Figura 6(a) exibe o diagrama de implementação do RAVEN na rede do Ifes. Com essa finalidade, desenvolveu-se uma ferramenta, como prova de conceito, que recebe as informações de tráfego (fluxo) de forma contínua e instantânea (i.e., logo que elas são geradas pelo *Zeek* + *FlowMeter*). Normalmente, essas informações são armazenadas em arquivos *.log*. Entretanto, visando otimizar a obtenção dessas informações por parte

da ferramenta desenvolvida, optou-se por salvar essas informações diretamente em um banco de dados *SQLite*, implementando uma *stream* de dados customizada integrada diretamente ao *Zeek*. O vetor de características dos fluxos são obtidos a partir da consulta ao banco de dados, sendo posteriormente pré-processados e realizada a devida seleção das *features*, conforme descrito na Seção 4. Posteriormente, essas *features* são encaminhadas e avaliadas pelo modelo multi-classe, descrito na Seção 4.3.3. O treinamento do modelo foi realizado de forma *offline*, salvo e carregado na inicialização do RAVEN. Finalmente, parte-se para a classificação dos fluxos de forma dinâmica, quase em tempo real, a medida em que as amostras são encaminhadas para classificação.

Visando avaliar o tempo de resposta, decorrido entre a geração do *log* de fluxo e a efetiva detecção pelo RAVEN, realizamos 2.560 varreduras com todas as categorias de varreduras treinadas, incluindo varreduras com configurações diferentes em relação ao conjunto de treino e teste. Essas varreduras foram misturadas ao tráfego normal do instituto, que ocorria simultaneamente durante essa avaliação. A Figura 6(b) apresenta o diagrama de caixa (*boxplot*) do tempo de resposta do RAVEN, sendo considerado nesse cálculo a diferença entre o *timestamp* da coleta do fluxo, parâmetro fornecido pelo *Zeek*, e o *timestamp* após a classificação da amostra. A fim de, obter medidas estatísticas robustas e confiáveis, calculou-se o tempo para a classificação de todos os fluxos considerados maliciosos. A ferramenta obteve as seguintes métricas de desempenho: Média de 6,42 s, Desvio Padrão de 0,69 s, Menor Valor de 5,39 s e Maior Valor de 8,81 s.

O desempenho do RAVEN em termos de tempo de resposta deve-se ao fato de que, durante os experimentos, o tempo que o Zeek leva para sumarizar e registrar dados sobre os fluxos é de 5 s. Esse intervalo é determinado por configuração da ferramenta, sendo que o valor padrão é de 5 s. Dessa forma, pode-se afirmar que o tempo de resposta do RAVEN, em sua implementação atual (não otimizada) fica em média igual a 6,42 - 5 = 1,42 s, podendo ser de menos de 1 s em alguns casos. Embora a avaliação desse parâmetro não tenha sido explorada neste trabalho, essa mudança implicaria em um aumento significativo no volume de *logs* gerados pela ferramenta, representando um *trade-off* entre tempo de resposta *versus* espaço de armazenamento e processamento.

6. Conclusão e Trabalhos Futuros

A principal contribuição deste artigo foi o projeto e implementação do RAVEN, um sistema capaz de identificar e classificar 8 diferentes tipos de varredura, com acurácia de 98,8% e com um tempo de resposta médio de 6,42 s, suficiente para permitir a rápida tomada de ações de mitigação, tais como bloqueio de tráfego e geração de *blacklists* de origens maliciosas. Estas ações são importantes na prevenção de futuros ataques mais danosos, como DDoS e intrusão. Os conjuntos de dados de treino e teste usados neste artigo, devidamente documentados, serão disponibilizados em domínio público. Como resultado adicional, foi quantificada a importância da ampliação dos atributos de fluxo.

Como trabalho futuro, pretende-se investigar a causa dos ataques *CONNECT* e *UDP* apresentarem amostras classificadas incorretamente. Avaliar a utilização de diferentes hiperparâmetros e processamento paralelo, de modo a melhorar a acurácia e o tempo de resposta. Além disso, pretende-se expandir os experimentos para caracterização, detecção e classificação de ataques de intrusão e integrá-los ao RAVEN da rede do Ifes. Finalmente, pretende-se também explorar a mitigação diretamente no plano de dados.

Agradecimentos

Este trabalho possui financiamento parcial da Fapes (#2023/ RWXSZ, #2022/ ZQX6, #2022/ NGKM5, #2021/ GL60J) e Fapesp/ MCTI/ CGI.br (#2020/ 05182-3). Os autores também gostariam de agradecer o apoio do Ifes *campus* Cachoeiro do Itapemirim/ES no desenvolvimento deste trabalho.

Referências

Araujo, A. M., Bergamini de Neira, A., and Nogueira, M. (2023). Autonomous machine learning for early bot detection in the internet of things. *Digital Communications and Networks*, 9(6):1301–1309.

Bhuyan, M. H., Bhattacharyya, D., and Kalita, J. (2011). Surveying Port Scans and Their Detection Methodologies. *The Computer Journal*, 54(10):1565–1581.

Bou-Harb, E., Debbabi, M., and Assi, C. (2013). A statistical approach for fingerprinting probing activities. In *2013 International Conference on Availability, Reliability and Security*, pages 21–30.

Cabaj, K. et al. (2018). Sdn-based mitigation of scanning attacks for the 5g internet of radio light system. In *Proceedings of the 13th Inter. Conference on Availability, Reliability and Security*, New York, NY, USA. Association for Computing Machinery.

Camacho, J. et al. (2019). Multivariate big data analysis for intrusion detection: 5 steps from the haystack to the needle. *Computers & Security*, 87:101603.

CERT.br (2023). Cert.br - estatística de incidentes notificados ao cert.br. `https://stats.cert.br/incidentes/`.

Devan, P. and Khare, N. (2020). An efficient xgboost–dnn-based classification model for network intrusion detection system. *Neural Comput. Appl.*, 32(16):12499–12514.

Do, E. H. and Gadepally, V. N. (2020). Classifying anomalies for network security. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2907–2911.

Griffioen, H. and Doerr, C. (2020). Discovering collaboration: Unveiling slow, distributed scanners based on common header field patterns. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9.

Habibi Lashkari, A. et al. (2017). Characterization of tor traffic using time based features. In *Int. Conference on Information Systems Security and Privacy*, pages 253–262.

Liu, J., Gao, Y., and Hu, F. (2021). A fast network intrusion detection system using adaptive synthetic oversampling and lightgbm. *Computers & Security*, 106:102289.

Mishra, P., Varadharajan, V., Tupakula, U., and Pilli, E. S. (2019). A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Communications Surveys & Tutorials*, 21(1):686–728.

Satheesh, N. et al. (2020). Flow-based anomaly intrusion detection using machine learning model with software defined networking for openflow network. *Microprocessors and Microsystems*, 79:103285.

Strom, B. E. et al. (2018). Mitre att&ck: Design and philosophy. In *Technical report*. The MITRE Corporation.