BG-IDPS: Detecção e Prevenção de Intrusões em Tempo Real em Switches eBPF com o Filtro de Pacotes Berkeley e a Metaheurística GRASP-FS

Diego Carvalho[1], **Vagner E. Quincozes** [2], **Silvio E. Quincozes** [3], **Juliano F. Kazienko** [1], **Carlos Raniery** [1]

[1] Universidade Federal de Santa Maria - UFSM

[2]Universidade Federal do Pampa - UNIPAMPA

[3]Universidade Federal de Uberlândia - UFU

dcardiego@ufsm.br, vagnerquincozes.aluno@unipampa.edu.br,

sequincozes@ufu.br, kazienko@redes.ufsm.br, csantos@inf.ufsm.br

Resumo. *Sistemas de detecção de intrusões modernos são comumente desenvolvidos com uso de algoritmos de aprendizado de máquina e seleção de atributos. No entanto, o custo computacional desses algoritmos limita a capacidade de resposta imediata às intrusões. Neste trabalho, é proposta uma arquitetura para a detecção e prevenção de intrusões em tempo real em switches eBPF a partir de modelos otimizados assincronamente através da metaheurística GRASP-FS. Como prova de conceito, um modelo é construído a partir de um computador e atualizado periodicamente em um switch eBPF. Os resultados obtidos revelam que a solução proposta é capaz de detectar e prevenir intrusões em tempo real com baixa sobrecarga para os cenários avaliados.*

Abstract. *Modern intrusion detection systems commonly employ machine learning algorithms and feature selection. However, their relatively high computational cost is an impediment to real-time intrusion prevention. In this work, we propose an architecture for real-time detection and prevention of intrusions within eBPF switches based on asynchronously optimized models through the GRASP-FS metaheuristics. As a proof of concept, we build a model from a computer that is periodically updated to an eBPF switch. Our results reveal that the proposed solution is capable of detecting and preventing intrusions in real-time with a low overhead in the assessed scenarios.*

1. Introdução

O avanço dos ataques cibernéticos nos últimos anos tem tomado proporções devastadoras em todo o mundo. Em 2022, mais de dois milhões de ucranianos foram vítimas de um ataque cibernético russo que teve como alvo as instalações elétricas do país [O'Neillarchive 2022]. No Brasil, não é diferente: conforme relatório mundial da Kaspersky, o país foi líder em ataques *phishing* em 2020 [Rodrigues 2022]. Ademais, foram registrados 270 ataques cibernéticos por empresa ao longo de 2021, o que representa um aumento de 31% em relação ao ano anterior [Accenture 2021].

A implementação de respostas em tempo real à crescente quantidade, variedade e evolução de estratégias de ataques cibernéticos é um dos principais desafios do desenvolvimento de Sistemas de Detecção e Prevenção de Intrusões, do inglês, *Intrusion Detection*

and Prevention Systems (IDPSs). Portanto, os IDPSs atuais mais sofisticados são tipicamente calcados em algoritmos de aprendizado de máquina para alcançar maior assertividade na decisão entre classificar uma atividade suspeita como legítima ou normal. Ademais, tais IDPSs costumam empregar algoritmos de pré-processamento, tais como aqueles que são destinados à seleção de atributos, ou no inglês, *Feature Selection* (FS). Com isso, modelos dinâmicos para a predição de ações maliciosas podem ser construídos com base nas informações mais representativas para cada perfil de ataque [Quincozes et al. 2021].

A implementação de algoritmos de alta complexidade computacional, como é o caso dos mais robustos algoritmos de aprendizado de máquina e seleção de atributos, requer o uso de componentes de *hardware* compatíveis. A análise de grande volumes de dados pode acarretar em desafios de *Big Data* (*i.e.,* grande volume, variedade e velocidade na geração de dados). Nesse contexto, as ferramentas comumente adotadas costumam explorar o uso de computação paralela e distribuída, de modo a ampliar-se o poder computacional disponível. Contudo, mesmo com um grande poder computacional, a análise de todo tráfego que passa por uma rede em tempo real (*i.e.,* antes da entrega do pacote ao destino) pode introduzir atrasos significativos aos serviços fornecidos pelo ambiente em questão. Isso acontece porque os pacotes são tradicionalmente encaminhados ao IDPS (*i.e.,* coletados e transmitidos para serem analisados externamente) para depois serem entregues ao destino [Mahdavisharif et al. 2021].

Em contraste com as ferramentas de análise de *Big Data*, uma outra alternativa consiste na implementação de IDPSs embarcados em *hardwares* de dispositivos programáveis, tais como Placa de Interface de Rede Inteligente, do inglês, *Smart Network Interface Cards* (SmartNICs). Tais dispositivos podem ser programados através da tecnologia de filtro de pacotes estendido, do inglês, *extended Berkeley Packet Filter* (eBPF) [Foundation 2022]. Ao passo que o uso de SmartNICs permite a análise de pacotes de redes em tempo real, sem a necessidade de encaminhamento de pacotes para outros dispositivos, as restrições computacionais desses dispositivos limitam a complexidade dos algoritmos implementados. Com isso, a implementação de algoritmos convencionais de aprendizado de máquina, do inglês, *Machine Learning* (ML) e seleção de atributos torna-se impraticável [Vieira et al. 2019].

Neste trabalho, é proposto um IDPS em tempo real baseado na metaheurística GRASP-FS [Quincozes et al. 2021] e na tecnologia eBPF [Foundation 2022], chamado *Berkeley and GRASP-FS Real-Time IDPS* (BG-IDPS). As principais contribuições deste trabalho consistem na extensão da metaheurística GRASP-FS para a otimização do modelo de detecção de intrusões de algoritmos de ML e na adaptação do algoritmo de ML *K-Nearest Neighbors* (KNN) através do filtro eBPF. Nesta arquitetura, a construção e otimização (*e.g.,* seleção de atributos) do modelo acontece paralelamente em um computador com capacidade computacional e de memória adequada, não impactando em termos de tempo a detecção de ataques. No *switch* programável com suporte a eBPF, apenas a etapa de classificação é executada. Para fins de avaliação, uma instância da arquitetura proposta e seus componentes foram simuladas. Os resultados obtidos revelam que a solução proposta é capaz de detectar e prevenir intrusões em tempo real com baixa sobrecarga para os cenários avaliados.

Este trabalho está organizado como segue. Na Seção 2, a fundamentação teórica é apresentada. Na Seção 3, o estado da arte no que tange à eBPF e prevenção de IDPS

em tempo real é revisado. Na Seção 4, é proposta a arquitetura BG-IDPS. Na seção 5, os resultados são apresentados e discutidos. Seção 6 as considerações finais são postuladas.

2. Fundamentação Teórica

Nesta Seção, são apresentados aspectos teóricos acerca da funcionalidade eBPF em switches (Seção 2.1), sistema de detecção e prevenção de intrusão (Seção 2.2) e seleção de atributos (Seção 2.3).

2.1. Extended Berkeley Packet Filter (eBPF)

O filtro de pacotes *Berkeley Packet Filter* (BPF) [McCanne and Jacobson 1993] foi proposto como uma solução para realizar a filtragem de pacotes no kernel de sistemas Unix BSD. Tal ferramenta consiste de um conjunto de instruções e uma máquina virtual para execução de programas escritos através de sua linguagem própria (*i.e., BPF)*. Devido à defasagem de instruções impostas por processadores modernos, novas instruções foram necessárias. Com isso, surgiu a versão estendida do BPF (eBPF) [Vieira et al. 2019].

A principal vantagem do eBPF é que não há necessidade de modificar o código-fonte ou carregar módulos existentes do kernel. Ao contrário, os aplicativos codificados podem ser injetados em pontos de rastreamento específicos [Deri et al. 2019]. Ademais, a tecnologia é capaz de proporcionar alto desempenho para redes, incluindo o balanceamento de carga em *data centers* modernos, dentre outras funções. O eBPF é capaz de extrair observabilidade de segurança de baixa granularidade, fornecendo *insights* para soluções de problemas de desempenho, aplicação preventiva e segurança em tempo de execução [Foundation 2022].

2.2. Sistemas de Detecção e Prevenção de Intrusão

Os Sistemas de Detecção e Prevenção de Intrusão, do inglês, *Intrusion Detection and Prevention Systems* (IDPSs) são ferramentas que auxiliam na identificação e mitigação de ataques às propriedades da segurança da informação.

Técnicas de detecção baseadas em anomalia são bem conhecidas na literatura, na qual padrões anormais são detectados, sem qualquer conhecimento prévio de ataques ou mesmo de suas assinaturas. Uma das principais vantagens desta abordagem consiste no potencial de detecção de comportamentos de atacantes que exploram novas formas de ataques – já que não é necessário o conhecimento prévio do mesmo. Por outro lado, uma das principais desvantagens consiste em que comportamentos anormais, mesmo que legítimos, possam ser considerados como ataques o que implica em um aumento no número de falsos positivos. Em geral, a identificação de anomalias pode ser alcançada através de métodos estatísticos ou de algoritmos de aprendizado de máquina por agrupamento de amostras de tráfego ou dados de aplicações similares [Quincozes et al. 2021].

Por outro lado, as técnicas baseadas em assinatura apresentam uma perspectiva diferente, concentrando-se na classificação de amostras de tráfego a partir de sua correspondência com padrões conhecidos em uma base para treinamento: as assinaturas. Essa técnica apresenta limitação para a detecção de ataques novos ou desconhecidos, ou seja, ataques para os quais não existem amostras na base de treinamento. No entanto, tal abordagem é conhecida por apresentar altas taxas de acertos para ataques conhecidos. Com isso, surge a importância da atualização periódica da base utilizada para treinamento e

de retreinamento constante quando da adoção dessas técnicas. Ademais, algoritmos de aprendizado de máquina classificadores podem ser categorizados dentro da técnica supervisionada, uma vez que operam sobre amostras anotadas as quais representam assinaturas de ataques e sua classe correspondente [Quincozes et al. 2021].

Para além da detecção de intrusões, é possível a implementação de medidas preventivas, as quais implicam na eliminação do tráfego malicioso. Quando um IDS tem a capacidade de detectar e também prevenir um ataque, este é denominado um IDPS. Uma alternativa para a implementação de IDPSs em tempo real consiste na utilização de *switches* programáveis, tais como os *switches* eBPF.

2.3. Seleção de Atributos

A seleção de atributos tem por finalidade a redução de dimensionalidade dos dados através do descarte daqueles atributos que são irrelevantes ou redundantes. Com isso, é esperado que após o processo de seleção de atributos os resultados sejam mais precisos e com tempo de processamento reduzido pelo descarte de informações de entrada desnecessárias.

Existem diferentes categorias de métodos de seleção de atributos. Os mais simples são os métodos baseados em *filtragem*. Tais métodos analisam os atributos individualmente com base em cálculos estatísticos, tais como a variação da entropia. Já os métodos *Embedded* são caracterizados por efetuar seleção de atributos durante o procedimento de classificação, onde o algoritmo de seleção está tipicamente embutido no algoritmo de classificação. Já os métodos baseados em *Wrapping* consistem em um meio termo, onde são utilizados algoritmos de aprendizado de máquina para avaliar os potenciais conjuntos de atributos que podem compor a solução. Nesse último método, o conjunto de atributos selecionado pode ser utilizado posteriormente para filtrar amostras sem a necessidade de um novo processo de seleção de atributos. É importante observar que métodos *wrapping* são mais efetivos que métodos de *filtragem*, pois em vez de apenas prever o valor do atributo com base em cômputos estatísticos, tal valor é medido diretamente a partir do resultado de testes envolvendo algoritmos classificadores.

O problema de seleção de atributos pode ser generalizado como um problema de otimização combinatória [Feo and Resende 1989], o qual pertence à classe *NP-Difícil* [Naghibi et al. 2013][Yusta 2009]. Na literatura, problemas dessa classe são comumente resolvidos com uso de metaheurísticas como o GRASP-FS [Quincozes et al. 2020, Quincozes et al. 2021] GRASP-FS é uma metaheurística híbrida que combina técnicas de *filtragem*, com a finalidade de reduzir o número de atributos a serem analisados posteriormente por um procedimento baseado em *wrapping* com suporte de algoritmos de ML.

3. Trabalhos Relacionados

Nesta seção, é apresentada uma revisão da literatura focada nos algoritmos para seleção de atributos baseada em metaheurísticas (Seção 3.1) e na detecção de intrusões baseada em eBPF (Seção 3.2).

3.1. Seleção de Atributos baseada em metaheurísticas

metaheurísticas são comumente usadas para resolver problemas de otimização combinatória. O problema de FS é, essencialmente, um desses problemas. Nas últimas

décadas, a FS baseada em métodos de busca local demonstrou ser eficaz em diferentes domínios. Em geral, pode-se classificar esses métodos em dois grupos de acordo com seu comportamento. O primeiro grupo é composto de métodos sequenciais, como SFFS e IFFS [Nakariyakul and Casasent 2009]. O segundo grupo é composto por métodos que podem ser analisados estatisticamente, mas se comportam heuristicamente, incorporando etapas não determinísticas, como GA, SA, Busca Tabu, ILS e GRASP [Esseghir 2010].

Yusta [Yusta 2009] demonstra que o GRASP pode superar os algoritmos SFFS, Busca Tabu, GA e algoritmos Memeticos na geração do melhor subconjunto de atributos para classificar amostras de diferentes bancos de dados (*i.e.,* Spambase, Waveform, Ionosphera, Vehicle, Wincosin e German) através do algoritmo classificador KNN. Posteriormente, Esseghir e Amir [Esseghir 2010] também aplicam o GRASP para FS, considerando alguns desses conjuntos de dados (*i.e.,* Ionosfera e SpamBase) e outros distintos (*i.e.,* Sonar, Audiologia e Arritmia). Esseghir e Amir usam Redes Neurais Artificiais e o algoritmo *Naive Bayes* como classificadores.

De fato, atualmente o uso de metaheurísticas já é considerado para FS na literatura [Yusta 2009][Esseghir 2010][Diez-Pastor et al. 2011][Díez-Pastor et al. 2014] [Kanakarajan and Muniasamy 2016]. No entanto, seu uso no domínio da detecção de intrusões foi pouco explorado até os últimos anos. Recentemente, com a proposta do GRASP-FS [Quincozes et al. 2020] , verificou-se que essa metaheurística se apresenta promissora para o domínio da detecção de intrusões.

3.2. Detecção de Intrusões baseadas em eBPF

Uma comparação de velocidade entre eBPF e uma solução sem essa funcionalidade é apresentada por [Bachl et al. 2021]. Os autores desenvolvem um IDS que é suportado tanto por um *host* quanto por roteadores e *switch* com suporte a eBPF. Os resultados indicam que a implementação eBPF é 20% mais rápida.

Do mesmo modo, [Wang and Chang 2022] projetam e implementam um sistema IDS para filtrar e verificar pacotes que chegam diretamente no kernel do Linux. O IDS é composto por duas partes que se complementam. A primeira é baseada em eBPF e executada no kernel do Linux. Ela é responsável por verificar e pré-descartar pacotes usando correspondências de padrões. A segunda parte é executada no espaço de usuário. Seu papel é examinar detalhadamente os pacotes deixados pela primeira parte.

Outros trabalhos propõem sistemas em tempo real usando eBPF. Por exemplo, um sistema baseado em eBPF capaz de monitorar a latência entre contêineres em tempo real é proposto por [Shiraishi et al. 2020]. Já [Van Tu et al. 2021] apresentam um método para monitorar o tempo de processamento de pacotes *Network Function Virtualization* (NFV) em tempo real. O método é construído usando eBPF para processar pacotes em alta velocidade. Para reduzir a sobrecarga no desempenho, algoritmos de amostragem e algoritmos de filtragem de eventos que filtram dados não importantes foram implementados. O método é executado dentro de máquinas virtuais, do inglês, *Virtual Machines* (VMs). Para fazer a ponte entre redes VMs e NFV, eles adotam o *Open vSwitch*. Uma limitação desse trabalho é que o método não é executado no *host* físico onde as VMs e NFV estão localizadas.

Outra solução prevê a situação de rede em tempo real [Zhang et al. 2021]. Nessa proposta, há dois módulos. O primeiro visa extrair dados de rede do Linux usando eBPF.

O segundo módulo é responsável por treinar os dados extraídos usando o modelo de rede neural LSTM e por prever a situação da rede por meio de simulação.

Ademais, existem trabalhos que usam eBPF em tempo real em outros contextos. Por exemplo, [Agman and Hendler 2021] propõem uma estrutura para detectar ameaças de *malware* em Android. Eles adotam o eBPF para monitorar eventos de aplicativos de usuário em tempo real. A proposta rastreia chamadas de API, de sistemas, funções de bibliotecas nativas e funções internas do kernel. Adicionalmente, novas assinaturas são desenvolvidas para detectar comportamentos anormais e capturar artefatos forenses. Os resultados mostram que quando um objeto suspeito é detectado a solução alerta em tempo real. Isso não incorre em sobrecarga significativa no tempo de execução.

Um estudo sobre as implicações de desempenho no filtro de pacotes com eBPF é proposto por [Scholz et al. 2018]. Os autores apresentam dois estudos de caso. O primeiro adota o *eXpress Data Path* (XDP) para processar o tráfego de entrada, enquanto que o segundo é usado para filtrar pacotes atuando no nível de *socket*. Os dois estudos de casos são implementados em nível de sistema (*i.e.*, no nível do sistema operacional).

Os autores de [Kostopoulos et al. 2021] propõem um esquema implementado via plano de dados programáveis para identificar o ataque de Negação de Serviço Distribuído, do inglês, *Denial of Service* (DDoS) *Water Torture* em servidores DNS. O referido esquema propõe algoritmos de ML para filtrar ataques nos resolvedores de DNS. Na solução, é implementado o classificador Naive Bayes para descartar requisições de DNS consideradas inválidas antes de qualquer recurso ser alocado para ela, enquanto apenas requisições válidas são enviadas para o espaço do usuário a fim de serem resolvidas. A solução aponta ainda que o XDP habilita a execução de programas eBPF no driver da interface de rede para cada pacote recebido antes de qualquer recurso de memória ser alocado para os pacotes recebidos, estabelecendo um caminho de dados de alta performance dentro do kernel do Linux. Desse modo, o autor destaca que as referidas vantagens tornam o XDP adequado para casos de uso como mitigação de ataques DDoS por exemplo.

3.3. Discussão

De acordo com a revisão da literatura realizada, tanto a adoção de metaheurísticas para a otimização de algoritmos da detecção de intrusões, quanto o emprego da tecnologia eBPF para análise de tráfego vêm sendo explorados. No entanto, o processamento por metaheurística envolve alto custo computacional e não é compatível com prevenção de intrusões em tempo real. Por outro lado, o processamento em *switches* eBPF é limitado por suas restrições computacionais e pode não prover o desempenho de detecção adequado. Uma abordagem que contemple a combinação desses dois artefatos em conjunto ainda não foi proposta.

4. Proposta

De modo a resolver o problema da detecção em tempo real com alta assertividade e velocidade, nesta seção é proposto um IDPS em tempo real baseado na metaheurística GRASP-FS [Quincozes et al. 2021] e na tecnologia eBPF [Foundation 2022], chamado *Berkeley and GRASP-FS Real-Time IDPS* (BG-IDPS). A arquitetura da proposta é ilustrada na Figura 1. Suas propriedades são as seguintes:

1. Detecção e prevenção de intrusões em tempo real (*i.e.,* embarcada em *switches*);

2. Processamento leve no módulo de detecção/prevenção (*i.e., switch*);
3. Aprendizado contínuo (*i.e.,* suporte à atualização periódica).

De modo a alcançar tais propriedades, o BG-IDPS concentra o processamento mais complexo — *i.e.,* seleção de atributos e construção de modelo — no componente de otimização de modelos baseado na metaheurística GRASP-FS, que é executado em um computador (treinador) com capacidade computacional e de memória adequada. Por outro lado, a detecção em tempo real (*i.e.,* baseada no modelo construído no treinador) acontece em um *switch* programável com suporte à eBPF.

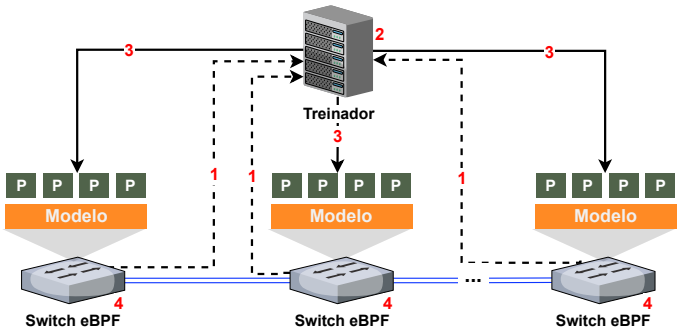

Figura 1. Arquitetura proposta.

A arquitetura proposta, que é exibida na Figura 1, é descrita a partir dos seguintes passos. No Passo 1, os dados atualizados sobre os fluxos de um swtich são enviados ao treinador. Por conseguinte, no Passo 2, o treinador seleciona um conjunto de parâmetros P otimizados para o classificador do switch. Já no Passo 3 os switches recebem instruções para a calibração dos seus classificadores. Por fim, o classificador interno do switch é calibrado no Passo 4. As seções seguintes descrevem os componentes da arquitetura proposta.

4.1. Componente Treinador

O componente treinador consiste em um computador com capacidade de memória e processamento a fim de suportar a execução de algoritmos de ML e seleção de atributos de modo a permitir a construção de modelos otimizados para a detecção de intrusões.

Tal componente usa de uma base de assinaturas contendo amostras de tráfego com anotações referentes às suas respectivas classes (*e.g.,* normal ou ataque). Especialmente para a geração do primeiro modelo, o processo de anotação pode ser manual ou substituído por um conjunto de dados já anotado disponível para esta finalidade (*datasets*). Posteriormente, havendo a disponibilidade de assinaturas com suas devidas anotações, a arquitetura se retroalimenta. Ou seja, o tráfego já classificado pelo *switch* eBPF será a nova entrada do componente treinador, conforme representado no Passo 1 da Figura 1.

Um elemento fundamental do componente treinador consiste no algoritmo que implementa a metaheurística GRASP-FS, ilustrada no Passo 2 da Figura 1, a qual foi estendida de [Quincozes et al. 2021] . Cabe destacar que uma contribuição deste trabalho consiste na extensão da saída do GRASP-FS que antes era limitada a uma *string* representando um subconjunto de atributos, para um mapa eBPF que descreve um modelo de detecção de intrusões e pode ser carregado no switch, conforme o Passo 3 da Figura 1.

O GRASP-FS consiste em uma abordagem híbrida para a seleção de atributos que combina técnicas baseada em *filtragem* com técnicas baseadas em *wrapping*, tendo como objetivo a redução do custo computacional em alcançar conjuntos de atributos eficientes. Basicamente, existem duas etapas no GRASP-FS, a saber: etapa de Construção e etapa de Busca Local, conforme denotado no Algoritmo 1.

Algoritmo 1: ALGORITMO GRASP-FS [QUINCOZES ET AL. 2020]

Entrada:
$Atributos_{Lista}$: Lista Total de Atributos
$MaxIteracoes$: Número máximo de iterações
$Tamanho$: Tamanho da seleção desejada
Saída:
Mod_O: Modelo otimizado com o melhor conjunto de atributos obtido

```
1  início
2      Mod_O ← ∅
3      Mod_O.acuracia ← 0
4      enquanto (i ≤ maxIteracoes) faça
5          Mod_INICIAL ← construir(Tamanho, Atributos_Lista)
6          Mod_VIZINHO ← buscaLocal(Mod_INICIAL, RCL_lista)
7          i ← i + 1
8          se Mod_VIZINHO.acuracia > Mod_INICIAL.acuracia então
9              Mod_O ← Mod_VIZINHO
10         fim
11         senão
12             Mod_O ← Mod_INICIAL
13         fim
14     fim
15 fim
16 retorna Mod_O
```

Na linha 5 do Algoritmo 1, soluções iniciais são construídas aleatoriamente a partir de um subconjunto de atributos obtidos por meio de um algoritmo de *filtragem*. Em seguida, na linha 6 do Algoritmo 1, são realizadas buscas locais iterativas de modo a otimizar soluções construídas anteriormente. Tais etapas de construção e busca local são detalhadas em [Quincozes et al. 2021]. A cada iteração, a melhor solução obtida entre as fases de construção e busca local é armazenada. Quando um critério de parada for alcançado (*e.g.,* número máximo de iterações ou tempo de execução), a melhor solução encontrada é usada como saída para o algoritmo GRASP-FS. Note que o GRASP-FS pode ser executado continuamente e novos modelos otimizados (Mod_O) podem ser instalados qualquer momento no *switch*. Dessa forma, é alcançada a melhoria contínua no aprendizado [Quincozes et al. 2020].

4.2. Componente Detector

O componente detector consiste em um *switch* programável com suporte à tecnologia eBPF[1]. Tal componente, ilustrado no Passo 4 da Figura 1, é responsável por receber modelos construídos pelo componente treinador e efetuar a análise de pacotes em tempo real. Com isso, é possível não só a detecção de intrusões mas também a prevenção das mesmas.

A partir deste dispositivo, propomos uma implementação simplificada baseada no algoritmo *K-Nearest Neighbors* (KNN) [Guo et al. 2003], o qual denominamos KNN leve, ou *Lightweight* KNN (L-KNN). O algoritmo é considerado leve porque adota

[1]Este dispositivo pode ser substituído por qualquer dispositivo com suporte à tecnologia eBPF, tal como placa de rede inteligente (*e.g.,* SmartNIC).

operações simplificadas que são suportadas pela linguagem eBPF. Tal adaptação, que é apresentada no Algoritmo 2, consiste em uma contribuição original deste trabalho.

Algoritmo 2: IMPLEMENTAÇÃO SIMPLIFICADA DO KNN (L-KNN)

Entrada: Um conjunto $amostras_{treino}$ com n assinaturas com m atributos cada, o número K de vizinhos mais próximos e uma amostra t cuja classe é desconhecida.
Saída: As K amostras vizinhas mais próximos de t.

```
1  início
2      KNN ← ∅ // inicialização dos K vizinhos mais próximos
3      para cada a ∈ amostras_treino faça
4          dist_a_t ← 0 // distância Manhattan entre a e t
5          numAtributos ← |t| // quantidade de atributos de t
6          enquanto i < numAtributos faça
7              dist_a_t ← |atributo_a[i] − atributo_t[i]| + dist_a_t
8              i++
9          fim
10         para cada k ∈ 1...K faça
11             se dist_a_t < dist_k_t então
12                 KNN_k ← a // a amostra a é a k^th mais próxima de t
13                 fim para // interrompe o laço e não afeta os k maiores
14             fim
15         fim
16     fim
17 fim
18 retorna KNN // as K amostras mais próximos de t
```

Basicamente, as entradas do Algoritmo 2 consistem em um conjunto de amostras com um conjunto restrito de atributos (providas pelo GRASP-FS) e um número fixo de vizinhos (*K*), os quais são usados para a classificação de amostras cuja classe é desconhecida (*t*). Nas linhas 3–16, acontecem iterações por todo o universo de amostras conhecidas. Especificamente, para cada atributo das amostras conhecidas, são subtraídos os valores da amostra desconhecida. Tais diferenças são somadas (linha 7).

Algoritmo 3: CLASSIFICAÇÃO DE AMOSTRAS DE TESTE PELO L-KNN

Entrada: Um conjunto das K amostras, cujas classes são conhecidas, mais próximas de t, cuja classe não é conhecida.
Saída: Classe atribuída à amostra de teste t.

```
1  início
2      para cada k ∈ 1...K faça
3          se classe_k = ataque então
4              ataque++
5          fim
6          senão
7              normal++
8          fim
9      fim
10     se ataque > normal então
11         classe ← ataque
12     fim
13     senão
14         classe ← normal
15     fim
16     retorna classe
17 fim
```

A partir da distância calculada, as *K* amostras conhecidas mais próximas, denominadas de vizinhos, são utilizadas pelo Algoritmo 3 para a classificação das amostras *t*. Para tal procedimento, adotou-se classe majoritária dentre os *K* vizinhos.

5. Resultados e Discussões

A fim de avaliar a arquitetura proposta, experimentos foram realizados. Na Seção 5.1, são descritos os cenários testados. Já na Seção na 5.2, os resultados obtidos são apresentados.

5.1. Cenário de Experimentação

Com a finalidade de simular o cenário de experimentação, foi utilizado ambiente de máquinas virtuais através do software de virtualização VirtualBox. Uma VM representa o módulo treinador e a outra exerce a função de módulo detector. Assim, deseja-se refletir, em menor escala, a arquitetura proposta na Seção 4.

A Máquina hospedeira, onde foram executadas os dois ambientes virtualizados (treinador e detector), possui a seguinte configuração: 12 *Gigabytes* (GB) de memória RAM, processador AMD FX (TM)-8350 com 8 Cores, 4 *Gigahertz* (GHz). Cada núcleo foi limitado para uso de até 75% de sua capacidade total. No ambiente, configurou-se uma máquina virtual (MV) com o sistema operacional (SO) convidado Linux CentOS 8 tanto para o módulo treinador quanto para ao módulo detector. A partir da virtualização do treinador, vários cenários de teste foram estabelecidos através da mudança de configurações de memória, algoritmos classificador e número de núcleos no SO convidado treinador. O SO Linux CentOS foi escolhido devido a característica de possuir as ferramentas necessárias para a instalação de programas eBPF no *driver* de interface de rede.

Para validar a solução proposta neste trabalho (Seção 4), utilizou-se o conjunto de dados NSL-KDD [Tavallaee et al. 2009], apropriado para algoritmos de classificação de *machine learning* (ML). Os atributos selecionados para esta prova de conceito são: o tipo de protocolo (*protocol_type*), serviço (*service*), status de erro ou conexão normal (*flag*), indicador de conexões de um host para ele mesmo (*land*) e número de fragmentos com erro (*wrong_fragment*). O conjunto de treino contém 125.973 amostras, enquanto que o de teste têm 22.544. O NSL-KDD foi utilizado para treinar o modelo e selecionar o ataque Smurf. As métricas consideradas neste contexto foram desempenho e o uso de memória. No experimento, transmitiu-se simultaneamente dois PCAPs para a interface de rede, totalizando 172.111 pacotes. O *dataset* NSL-KDD dispõe tanto de amostras em formato textual, compreensível para algoritmos de ML, quanto PCAPS.

Para o cômputo das métricas relacionadas ao desempenho de detecção, utilizou-se dos dados disponíveis para treinamento e teste de algoritmos de ML, abrangendo todas as classes de ataques contidas no NSL-KDD, as quais se dividem em: (i) *Denial-of-Service* (DoS): negação de serviço, tais como, SYN flood; (ii) *Remote-to-Local* (R2L): acesso não autorizado de uma máquina remota, tais como, ataques de força bruta; (iii) *User-to-Root* (U2R): acesso não autorizado a privilégios de superusuário local (*root*), tais como, ataques "buffer overflow"; e (iv) (*Probing*): sondagens, tais como, varredura de portas.

Já para os experimentos direcionados ao cômputo do tempo de execução, utilizou-se padrões do ataque *Denial-of-Service* (DoS) do tipo Smurf. Para tais experimentos, utilizou-se da suíte de utilitários conhecida como TCPReplay [2], que permite reproduzir tráfegos de rede. Optou-se pelo uso dessa ferramenta pela possibilidade de enviar PCAPS para a interface de rede e, assim, analisar-se o desempenho em termos de tempo de processamento em tempo real. Nas experimentações, considerou-se o envio de dois PCAPS: um

[2]Diponível em:`https://tcpreplay.appneta.com/wiki/overview.html`

normal e outro de ataque. Em primeiro momento, foram consideradas como métricas o tempo de envio e processamento de pacotes. As métricas foram avaliadas de três formas: considerando (i) um classificador com filtro eBPF, (ii) um classificador desenvolvido na linguagem de programação Python, e (iii) sem nenhum classificador.

Os experimentos foram realizados com diferentes configurações de memória e processador, a saber: 256, 512, 4096 e 8192 MBs de memória, e com 1, 2 e 4 processadores. Para mensurar sobre a métrica tempo de processamento, executou-se um teste com o classificador KNN eBPF e outro sem a imlplementação de classificadores.

5.2. Resultados

5.2.1. Desempenho de Detecção

De modo a executar um teste de sanidade, comparou-se a implementação do algoritmo L-KNN em eBPF com uma implementação usando a linguagem de programação Python. De acordo com os resultados ilustrados na Figura 2, ambas as implementações do algoritmo L-KNN obtiveram resultados consistentes (idênticos). Em particular, tanto a acurácia obtida quanto a F1-Score foram de 86%. Já para a métrica recall, alcançou-se um desempenho de 80%. A precisão foi de 92%. Com isso, a implementação em linguagem eBPF pode ser considerada consistente.

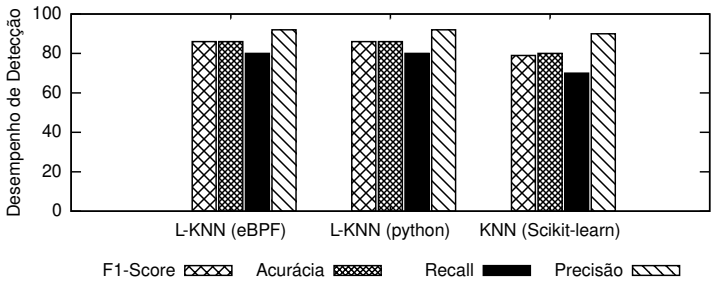

Figura 2. Resultados comparativos entre a implementação L-KNN e KNN.

De modo a comparar os resultados obtidos com o algoritmo L-KNN com o algoritmo KNN tradicional, foi utilizada a implementação desse algoritmo provida pela biblioteca Scikit-learn. Os resultados do L-KNN foram superiores em todas as métricas. Conforme a Figura 2, quando usada a biblioteca Scikit-learn, houve uma perda de desempenho de 7% na F1-Score. As métricas de acurácia, recall e precisão também foram afetadas negativamente com essa biblioteca em, respectivamente, 6%, 10% e 2%.

Portanto, a implementação L-KNN apresenta-se promissora para a detecção com uma assertividade razoável, considerando todos os ataques presentes no dataset NSL-KDD. Destaca-se ainda, que foram realizados também experimentos considerando o processamento embarcado, em *switch* eBPF, para tipos específicos de ataques, a saber: DoS Smurf. Os resultados obtidos foram ainda superiores. A F1-Score e Acurácia foram 98%. A recall alcançada foi de 100%, enquanto que a precisão foi de 95%.

5.2.2. Desempenho de Tempo

A fim de avaliar o desempenho do tempo de execução, realizaram-se experimentações através de diferentes plataformas e cenários.

Primeiramente, o tempo de execução do L-KNN foi comparado, na Figura 3, com o tempo obtido pela implementação do KNN disponível na biblioteca Weka [Witten et al. 2011]. Para este experimento foram consideradas os 5 atributos apontados na Seção 5.1. Como resultado, o L-KNN foi executado em apenas 54% do tempo total necessário para processar as 2.534 amostras pelo KNN do Weka.

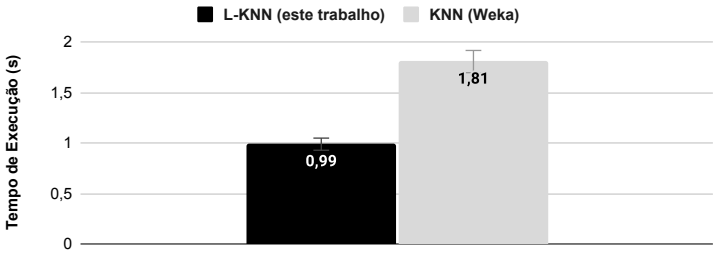

Figura 3. Tempo de execução entre o L-KNN (em java) e KNN (Weka).

A fim de avaliar o desempenho do tempo de execução, realizaram-se experimentações considerando treze cenários. Para tais experimentos, o GRASP-FS foi configurado para selecionar apenas dois atributos do *dataset*. A saída da seleção de atributos foi um modelo composto pelos atributos *protocol_type* e *service*. Os resultados das métricas tempo de processamento e pacotes por segundos são apresentados na Tabela 1.

Tabela 1. Tempo de processamento e vazão para os cenário avaliados.

Cenário	Configuração			Tempo de Processamento	Pacotes por Segundos
Id	**Memória**	**Processadores**	**Classificador**	**Segundos (s)**	**Megabytes por Segundo (Mbps)**
1	256	2	Sem classificador	81,52	28,81
2	8192	4	Sem classificador	81,52	28,81
3	512	2	Sem classificador	81,54	28,79
4	4096	2	Sem classificador	81,54	28,81
5	4096	2	KNN-eBPF	81,57	28,81
6	512	2	KNN-eBPF	81,61	28,76
7	8192	2	Sem classificador	81,62	28,77
8	8192	2	KNN-eBPF	81,63	28,78
9	8192	4	KNN-eBPF	81,63	28,79
10	256	2	KNN-eBPF	81,78	28,69
11	512	1	Sem classificador	82,07	28,53
12	512	1	KNN-eBPF	82,31	28,40
13	8192	4	KNN-Python	87,11	25,94

Em síntese, os cenários 1-4 apresentam os melhores resultados em tempo de processamento (*i.e.,* 81,52 e 81,54 segundos). Os cenários que adotam o classificador L-KNN em eBPF apresentam sobrecarga relativamente baixa (*i.e.,* 81,57 e 81,61 segundos). A implementação L-KNN em Python obteve o maior tempo de processamento, com 87,11 segundos. A quantidade de memória e processadores não interferiu significativamente no tempo de processamento. A maior disponibilidade de processadores não se refletiu em ganhos de tempo. Em relação aos pacotes por segundos, nota-se que a implementação

do classificador L-KNN em Python obteve a menor vazão (*i.e.,* 25,94 Mbps). Não foram observadas sobrecarga significativa com o uso do L-KNN em eBPF.

6. Conclusão

Neste trabalho, apresentou-se uma arquitetura para a detecção de prevenção de intrusão em tempo real baseada na metaheurística GRASP-FS e Switches eBPF chamada *Berkeley and GRASP-FS Real-Time IDPS* (BG-TIDPS). A arquitetura apresenta dois principais componentes: (i) treinador, que constrói modelos otimizados para a detecção de intrusão, e (ii) detector, que analisa pacotes em tempo real e detecta/previne intrusões em *switches*. Uma contribuição importante deste trabalho consiste na adaptação do algoritmo de ML *K-Nearest Neighbors* (KNN) chamada L-KNN.

Os experimentos revelam que a arquitetura é promissora. Do ponto de vista do tempo de execução, o classificador L-KNN leva metade do tempo médio para processar amostras de tráfego em relação a versão do KNN disponível na biblioteca Weka. O L-KNN levou 54% do tempo levado pelo KNN Scikit-learn, revelando maior eficiência por parte do algoritmo adaptado para este trabalho L-KNN na detecção de ataques.

Como trabalhos futuros pretende-se utilizar outros algoritmos classificadores, tais como o J48. Tais algoritmos são conhecidos por serem ainda mais rápidos.

Referências

Accenture (2021). O estágio atual da resiliência cibernética. `https://www.accenture.com/br-pt/insights/security/invest-cyber-resilience`. Acessed: 13-06-2022.

Agman, Y. and Hendler, D. (2021). BPFroid: Robust Real Time Android Malware Detection Framework. *arXiv preprint arXiv:2105.14344*.

Bachl, M., Fabini, J., and Zseby, T. (2021). A flow-based IDS using Machine Learning in eBPF. *arXiv preprint arXiv:2102.09980*.

Deri, L., Sabella, S., Mainardi, S., Degano, P., and Zunino, R. (2019). Combining System Visibility and Security Using eBPF. In *ITASEC*.

Díez-Pastor, J.-F., García-Osorio, C., and Rodríguez, J. J. (2014). Tree ensemble construction using a GRASP-based heuristic and annealed randomness. *Information Fusion*, 20:189–202.

Diez-Pastor, J. F., García-Osorio, C., Rodríguez, J. J., and Bustillo, A. (2011). GRASP Forest: A New Ensemble Method for Trees. In *Int. Workshop on Multiple Classifier Systems*, pages 66–75. Springer.

Esseghir, M. A. (2010). Effective wrapper-filter hybridization through GRASP schemata. In *Feature Selection in Data Mining*, pages 45–54.

Feo, T. A. and Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71.

Foundation, T. L. (2022). eBPF . https://ebpf.io/. Acessado: em Junho/2022.

Guo, G., Wang, H., Bell, D., Bi, Y., and Greer, K. (2003). KNN model-based approach in classification. In *OTM Confederated International Conferences*, pages 986–996. Springer.

Kanakarajan, N. K. and Muniasamy, K. (2016). Improving the Accuracy of Intrusion Detection Using GAR-Forest with Feature Selection. In *Proceedings of the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA)*, pages 539–547. Springer.

Kostopoulos, N., Korentis, S., Kalogeras, D., and Maglaris, V. (2021). Mitigation of DNS Water Torture Attacks within the Data Plane via XDP-Based Naive Bayes Classifiers. In *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, pages 133–139.

Mahdavisharif, M., Jamali, S., and Fotohi, R. (2021). Big Data-Aware Intrusion Detection System in Communication Networks: a Deep Learning Approach. *Journal of Grid Computing*, 19(4):1–28.

McCanne, S. and Jacobson, V. (1993). The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX winter*, volume 46.

Naghibi, T., Hoffmann, S., and Pfister, B. (2013). Convex approximation of the NP-hard search problem in feature subset selection. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3273–3277. IEEE.

Nakariyakul, S. and Casasent, D. P. (2009). An improvement on floating search algorithms for feature subset selection. *Pattern Recognition*, 42(9):1932–1940.

O'Neillarchive, P. H. (2022). Russian hackers tried to bring down Ukraine's power grid to help the invasion. https://www.technologyreview.com/2022/04/12/1049586/russian-hackers-tried-to-bring-down-ukraines-power-grid-to-help-the-invasion/. Acessed: 05-09-2022.

Quincozes, S. E., Mossé, D., Passos, D., Albuquerque, C., Ochi, L. S., and dos Santos, V. F. (2021). On the Performance of GRASP-Based Feature Selection for CPS Intrusion Detection. *IEEE Trans. on Net. and Service Management*.

Quincozes, S. E., Passos, D., Albuquerque, C., Ochi, L. S., and Mossé, D. (2020). GRASP-based Feature Selection for Intrusion Detection in CPS Perception Layer. In *2020 4th Conference on cloud and internet of things (CIoT)*, pages 41–48. IEEE.

Rodrigues, C. (2022). Preparado para o próximo ciberataque? `https://revista.consumidormoderno.com.br/preparado-para-proximo-ciberataque`. 13-06-2022.

Scholz, D., Raumer, D., Emmerich, P., Kurtz, A., Lesiak, K., and Carle, G. (2018). Performance Implications of Packet Filtering with Linux eBPF. In *International Teletraffic Congress*, pages 209–217.

Shiraishi, T., Noro, M., Kondo, R., Takano, Y., and Oguchi, N. (2020). Real-time Monitoring System for Container Networks in the Era of Microservices. In *2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 161–166.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on comp. intelligence for security and defense applications*, pages 1–6.

Van Tu, N., Yoo, J.-H., and Hong, J. W.-K. (2021). PPTMon: Real-Time and Fine-Grained Packet Processing Time Monitoring in Virtual Network Functions. *IEEE TNSM*, 18(4):4324–4336.

Vieira, M. A., Pacífico, R. D., Castanho, M. S., Santos, E. R., Júnior, E. P. C., and Vieira, L. F. (2019). Processamento Rápido de Pacotes com eBPF e XDP. *Sociedade Brasileira de Computação*.

Wang, S.-Y. and Chang, J.-C. (2022). Design and implementation of an intrusion detection system by using Extended BPF in the Linux kernel. *Journal of Network and Computer Applications*, 198:103283.

Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann Publishers (Elsevier), 3 edition.

Yusta, S. C. (2009). Different metaheuristic strategies to solve the feature selection problem. *Pattern Recognition Letters*, 30(5):525–534.

Zhang, X., Liu, Z., and Bai, J. (2021). Linux Network Situation Prediction Model Based on eBPF and LSTM. In *2021 16th Inter. Confer. on Intelligent Systems and Knowledge Engineering*, pages 551–556.