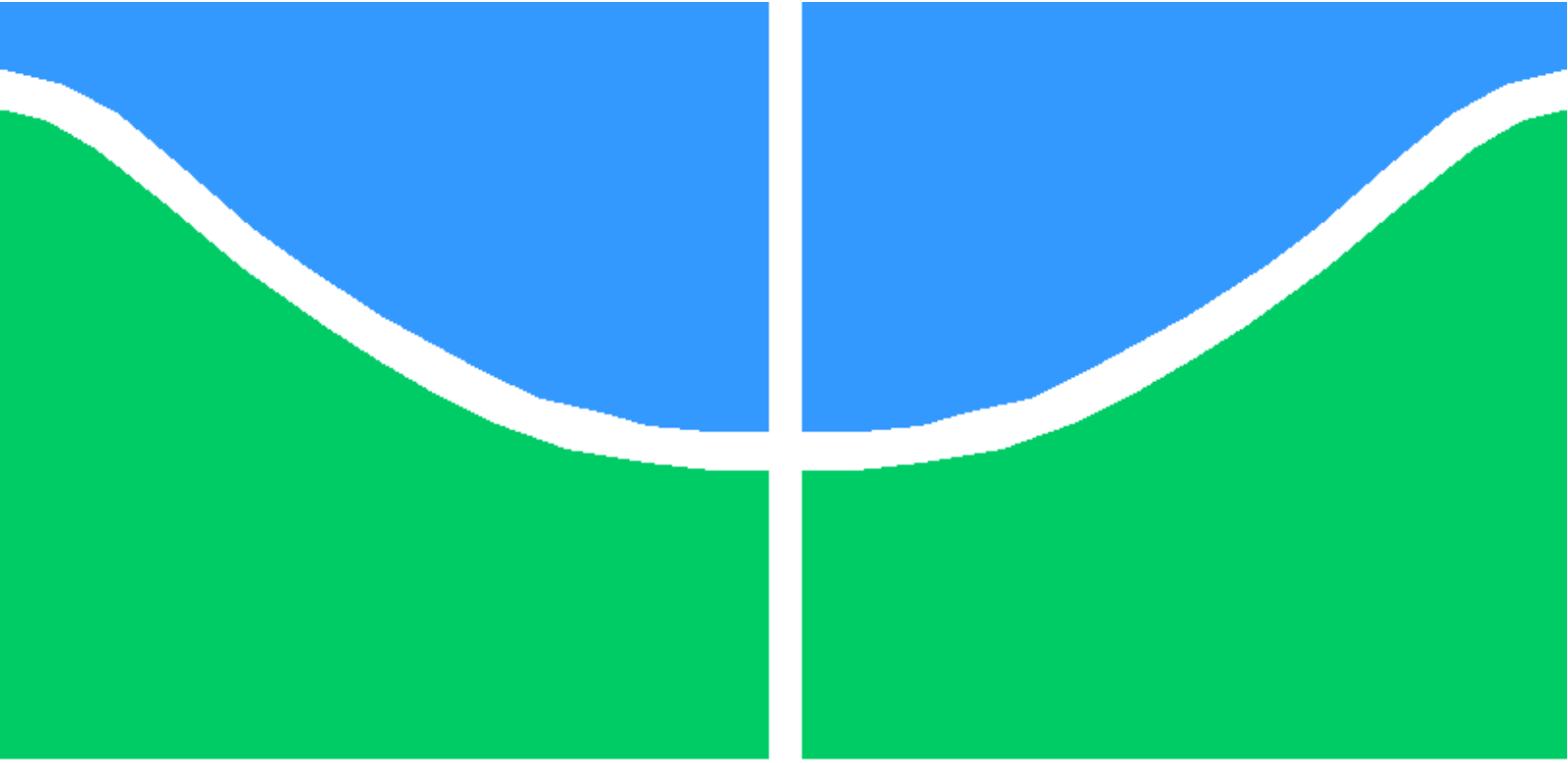


Universidade de Brasília - UnB

Faculdade de Tecnologia

Engenharia de Redes de Comunicação

Sistemas de Detecção de Botnets: Uma Análise Comparativa

Daniel Pereira Gonçalves

Lucas Alexandre Carvalho Chaves

Orientador: Alexandre Solon Nery

Brasília, DF

2020

Daniel Pereira Gonçalves
Lucas Alexandre Carvalho Chaves

Sistemas de Detecção de Botnets: Uma Análise Comparativa

Monografia submetida ao curso de graduação em Engenharia de Redes de Comunicação da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Redes de Comunicação.

Universidade de Brasília - UnB

Faculdade de Tecnologia

Orientador: Alexandre Solon Nery

Brasília, DF
2020

Daniel Pereira Gonçalves
Lucas Alexandre Carvalho Chaves
Sistemas de Detecção de Botnets: Uma Análise Comparativa
Brasília - DF, 2020
Orientador: Alexandre Solon Nery

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade de Tecnologia

1. botnet. 2. IDS. 3. fluxos de rede. 4. aprendizado de máquina. 5. grafos.

Daniel Pereira Gonçalves
Lucas Alexandre Carvalho Chaves

Sistemas de Detecção de Botnets: Uma Análise Comparativa

Monografia submetida ao curso de graduação em Engenharia de Redes de Comunicação da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Redes de Comunicação.

Trabalho aprovado. Brasília, DF, 16 de outubro de 2020:

D. Sc. Alexandre Solon Nery
Orientador

Ph. D. Rafael Timóteo de Sousa Júnior

D. Sc. Georges Daniel Amvame Nze

Brasília, DF
2020

Resumo

Com o rápido crescimento do número de dispositivos conectados à internet, cada vez mais as botnets representam um grande perigo para a segurança das informações e para o funcionamento legítimo de serviços e aplicações em nuvem. Se faz necessário o estudo de formas para detecção automática de uma ou várias botnets por meio da observação das informações referentes ao tráfego de rede. Na literatura, é possível encontrar diferentes métodos de análise e detecção de bots utilizando aprendizado de máquina. Neste trabalho, foi realizado um estudo de comparação entre a análise por meio de *features* de fluxos de rede e por meio de *features* de grafos através de algoritmos de redes neurais e SOM (*self-organizing maps*). Os *datasets* CTU-13, CIC 2017 e CIC 2018 são conjuntos de dados disponíveis publicamente com as características desejáveis para este tipo de análise e, por isso, foram utilizados para treinamento e teste dos modelos contruídos. Os resultados mostram que a análise do tráfego de rede por meio de grafos proporciona uma melhor qualidade na classificação dos bots e uma menor dependência das características específicas de cada tipo de botnet.

Palavras-chaves: botnet, IDS, fluxos de rede, aprendizado de máquina, grafos

Abstract

With the fast growth in the number of devices connected to the internet, botnets represent a great danger to the security of information and to the legitimate functioning of cloud services and applications. Therefore, it is necessary studying ways to detect a botnet by observing the information related to the network traffic. In the literature, it is possible to find different methods of analyzing and detecting bots using machine learning. In this work, it was conducted a comparative study between an analysis by network flows and an approach based on graph features through neural networks and SOM (self-organizing maps). The CTU-13, CIC 2017 and CIC 2018 are publicly available datasets with desirable characteristics for this type of analysis and, therefore, were used for training and testing the models built. The results show that the analysis of network traffic by the graphs method offers a better quality in the classification of bots and less dependence on the specific characteristics of each type of botnet.

Key-words: botnet, IDS, network flows, machine learning, graphs.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IoT	Internet das Coisas (Internet of Things)
IDC	International Data Corporation
DDOS	Ataque de Negação de Serviço Distribuído (Distributed Denial of Service)
IDS	Sistema de Detecção de Intrusão (Intrusion Detection System)
C&C	Unidade Central de Comando e Controle (Command and Control)
HTTP	Protocolo de Transferência de Hypertextos (Hypertext Transfer Protocol)
IRC	Internet Relay Chat
P2P	Ponto a Ponto (Peer-to-peer)
DNS	Servidor de Nome de Domínio (Domain Name Server)
RAM	Memória de Acesso Aleatório (Random Access Memory)
IP	Protocolo de Internet (Internet Protocol)
HTML	Linguagem de Marcação de Hypertexto (Hypertext Markup Language)
SOM	Mapa auto-organizável (Self-organizing map)
VP	Verdadeiro Positivo
VN	Verdadeiro Negativo
FP	Falso Positivo
FN	Falso Negativo
CIC	Canadian Institute for Cybersecurity
CTU	Czech Technical University
TCP	Protocolo de Controle de Transmissão (Transmission Control Protocol)
UDP	User Datagram Protocol
SVM	Máquina de vetores de suporte (Support Vector Machine)

ANN	Redes neurais artificiais (Aritificial neural networks)
API	Interface de Programação de Aplicações (Application Programming Interface)
MLP	Multilayer Perceptron
FNN	Feed-forward Neural Network
Tanh	Tangente hiperbólica
ReLU	Retificador (Rectified Linear Units)
BMU	Best Matching Unit

Sumário

1 Problema e Objetivos

1.1 Contexto

O número de dispositivos conectados à internet continua crescendo a uma alta velocidade. Além disso, a quantidade de ataques cibernéticos também têm aumentado significativamente a cada ano que passa. No ano de 2019, o Brasil sofreu 15 bilhões de tentativas de ataques cibernéticos em um período de três meses [1]. Foi detectada a utilização de técnicas de ataque relativamente antigas, como as usadas no ransonware Wannacry (2017). Essas informações indicam que a recorrência destes tipos de ataques podem ser resultado da falta de correções em sistemas e programas que tem como fim evitá-los.

O número de dispositivos que fazem parte da Internet das Coisas (IoT) aumentou significativamente nos últimos anos e a tendência é continuar crescendo. Segundo a *International Data Corporation* (IDC), em 2025 haverá cerca de 41,6 bilhões de dispositivos IoT [2]. Como o nível de segurança destes dispositivos não tem acompanhado a velocidade de seu crescimento, a quantidade de ataques e a descoberta de vulnerabilidades também tendem a crescer. Pode-se citar como exemplo o surgimento de novas botnets que possuem como alvo dispositivos IoT, como a Mirai e suas variantes [3].

1.2 Problema e relevância

Com a expansão de dispositivos vulneráveis conectados à internet, a quantidade de ataques e a disseminação de *malwares* pela rede tende a aumentar. Isso facilita e propicia a criação de botnets: conjunto de *malwares* instalados em equipamentos que possuem acesso à internet com o objetivo de assumir o controle desses dispositivos, formando uma "rede zumbi". Cada um dos equipamentos infectados executa ações determinadas pelo atacante, podendo ser utilizado para diversos tipos de ataques, por exemplo, um ataque de negação de serviço distribuído (DDOS), acesso a informações confidenciais por meio de registro de teclas pressionadas (*keylogging*) ou disseminação de um ransomware - como ocorrido com o Nemty ransomware [46].

Dessa forma, pode-se perceber que as botnets representam um grande perigo, pois dispõem de uma variedade de ataques distintos e consequentemente, apresentando características distintas. Devido a isso, não é uma tarefa fácil a identificação desses softwares maliciosos e geralmente quando são descobertos, já é tarde demais. Uma maneira de tentar resolver esse problema seria pela análise dos pacotes que trafegam pela rede. Assim,

conseguindo-se identificar uma botnet, pode-se evitar diversos problemas que poderiam ser causadas pelo *malware*.

1.3 Objetivos gerais

Sabendo-se da grande capacidade destrutiva que uma botnet possui, o principal objetivo do trabalho é o desenvolvimento de um Sistema de Detecção de Intrusão (IDS), baseado em técnicas de aprendizado de máquina, que seja capaz de identificar de forma eficiente a presença de botnets na rede. Após o treinamento, o IDS deve poder diferenciar tráfego benigno de tráfego de *malware* com alta acurácia.

1.4 Objetivos específicos

A fim de alcançar o objetivo geral, os seguintes objetivos específicos foram definidos:

- Encontrar um ou mais *dataset*s realísticos e robustos, por meio dos quais seja possível realizar o treinamento do IDS.

- Selecionar *features* que sejam capazes de caracterizar o tráfego de botnets.

- Testar e analisar a eficiência de diferentes técnicas de aprendizado de máquina para a detecção de *malwares*.

- Avaliar a eficiência do IDS proposto para diferentes tipos de botnets e variações existentes.

1.5 Justificativa

Dada a grande quantidade de problemas relacionados a segurança emergindo, é de interesse de qualquer administrador de redes proteger seus ativos dos riscos dos diversos tipos de ataques realizados. As variações para cada *malware* evoluem rapidamente com os avanços na indústria de hardware, software e a criação de algoritmos cada vez mais sofisticados; logo, os sistemas de detecção devem ser capazes de acompanhar essa velocidade. Ataques cibernéticos são sérios problemas de segurança que demandam métodos de detecção robustos e flexíveis.

Uma vez que os métodos de detecção convencionais se baseiam no conhecimento sobre os ataques já existentes, não são úteis contra variações ou novos tipos de ataques. Portanto, um sistema de detecção que seja capaz de identificar ataques nunca vistos é extremamente útil no combate a *malwares*. Técnicas de aprendizado de máquina se enquadram justamente nesse perfil, pois ao invés de buscarem por um padrão específico e

fechado são treinadas para identificar padrões genéricos que podem ser observados mesmo quando variam, desde que a variação não seja muito grande.

1.6 Organização

A seção II estabelece toda a fundamentação teórica necessária para o entendimento do trabalho, detalhando aspectos como os tipos de botnet, suas variações e a capacidade que esse tipo de rede possui. Além disso, também são mostradas as possíveis técnicas utilizadas para detecção de *malwares* e, consequentemente, de botnets. A seção III detalha e metodologia que foi utilizada para a construção dos modelos propostos. A seção IV apresenta uma abordagem utilizando redes neurais em uma análise baseada em fluxos de tráfego de rede. Por fim, a seção V desenvolve uma abordagem de detecção baseada em grafos que representam os nós da rede e suas respectivas conexões.

2 Fundamentação Teórica

2.1 Botnet

Apesar de existirem vários tipos de botnet, é possível defini-las de uma forma bastante simplificada como uma série de computadores conectados e coordenados para executar uma tarefa [4], permitindo que o atacante (denominado botmaster) tenha a capacidade de acessar informações das vítimas ou tomar o controle dos dispositivos usados.

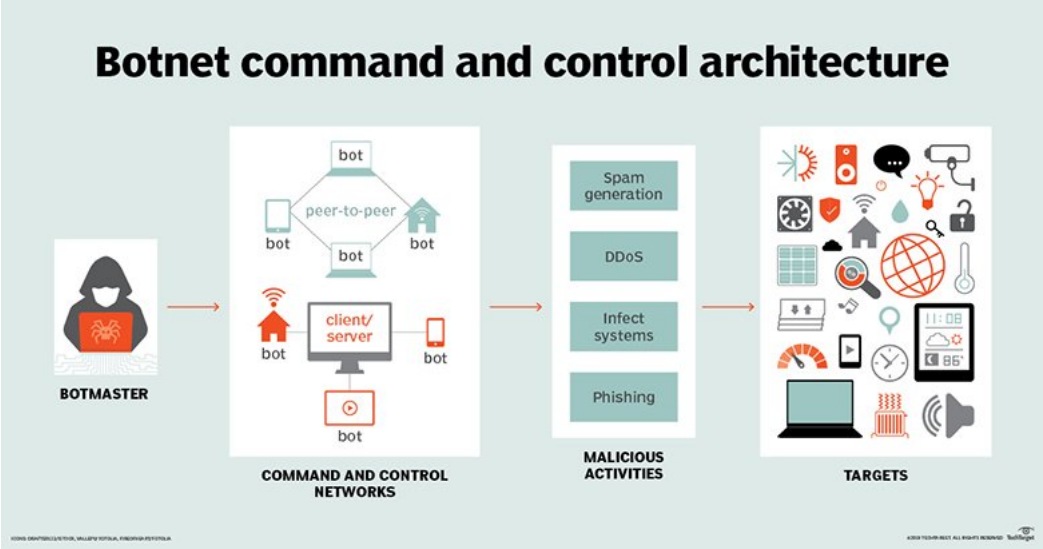


Figura 1 – Arquitetura de uma botnet centralizada
Fonte: Rouse (2019) [5]

2.2 Tipos de botnet

As botnets podem possuir diferentes arquiteturas: centralizada ou descentralizada. Em uma botnet centralizada, todos os bots ou zumbis (hospedeiros infectados) se comunicam com uma unidade central de comando e controle (C&C). Este tipo de topologia facilita a identificação da botnet, pois, caso a unidade de comando e controle deixe de funcionar, os bots não receberão mais ordens nem enviarão mais informações e, consequentemente, a botnet perde o sentido. Já em uma botnet descentralizada, não existe uma ponto único centralizado. Todos os bots agem como um C&C, tornando a botnet mais robusta e mais difícil de neutralizar.

O processo de criação de uma botnet centralizada pode ser descrito, simplificadamente, da seguinte maneira: um *hacker* constrói ou utiliza uma ferramenta de *exploit* para verificar as vulnerabilidades de uma rede qualquer e seus ativos (dispositivos IoT, com-

putadores, roteadores, etc). Tendo conhecimento das vulnerabilidades, as explora para se conectar aos dispositivos da rede escaneada. Conseguindo infectar os dispositivos, cada *malware* inserido irá forçar os hospedeiros a se conectarem ao C&C. A partir daí, o botmaster terá controle sobre os hospedeiros e poderá realizar uma série de ataques com diferentes propósitos.

Tendo uma botnet se espalhado em larga escala, interromper seu funcionamento por completo é extremamente difícil. Pode-se citar o nome de algumas que estão ativas atualmente, como por exemplo Kraken, Coficker, Cutwail, Storm, Emotet, Gucci e Ares. Além dos ataques focados às máquinas de uso pessoal ou servidores de grandes corporações, o grande número de dispositivos IoT emergindo e as vulnerabilidades que estes dipositivos possuem contribuem para a disseminação e criação de novas botnets.

2.3 Protocolos utilizados

A comunicação entre bots ou entre o C&C e os dispositivos infectados pode ser realizada por meio de diferentes protocolos, como o Hypertext Transfer Protocol (HTTP), Internet Relay Chat (IRC), Telnet, Peer-to-peer (P2P), Domain Name Server (DNS), entre outros. A utilização de cada um destes protocolos irá influenciar no tipo da botnet e em suas características. Além disso, dependendo do protocolo escolhido, a botnet pode ser mais facilmente detectada ou dificultar sua detecção.

2.4 Principais usos

Bots não foram inicialmente projetados para objetivos maliciosos. Seu propósito era realizar tarefas automatizadas como distribuir arquivos, realizar gerenciamento de canais ou responder perguntas em canais IRC [6]. Dada sua evolução e a grande gama de usos possíveis, passaram a ser utilizados para fins não benignos, como os seguintes:

- **Acesso indevido à informações:** Botnets podem ser usadas para adquirir números de cartões de crédito, credenciais de login, códigos de ativação de produtos, entre outros.
- **Monitoramento de tráfego:** Uma botnet também pode ser utilizada para monitorar o tráfego dos hosts infectados. Esse monitoramento pode ter como objetivo entender os padrões da rede ou realizar acesso a informações por meio de técnicas como keylogging.
- **Fraude:** As máquinas infectadas podem ser usadas para clicar em banners de anúncios em websites, falsamente aumentando o número de visitas.

- **Armazenamento:** - Máquinas infectadas podem ser usadas para armazenar conteúdo ilegal, softwares de terceiros ou mídia pirata, dificultando o rastreio do proprietário. Podem também armazenar e distribuir novas versões da botnet, com mais funcionalidades e mecanismos para burlar sistemas de detecção.

- **Processamento:** - Os atacantes podem utilizar as capacidades físicas dos infectados, se aproveitando do poder de processamento e Memória de Acesso Aleatório (RAM) para rodar algoritmos que os beneficiem. Um exemplo é o uso de zumbis para mineração de bitcoins.

2.5 Ataques

Uma botnet pode ser utilizada para diferentes tipos de ataques, sejam eles para o acesso a informações ou mesmo para deixar um servidor indisponível. A seguir são explicados ataques possíveis de serem realizados por meio de uma botnet:

- **Ramsonware:** Esse é um tipo de *malware* que impede os usuários de acessarem seu sistema ou arquivos pessoais e exige o pagamento do resgate (ransom) para liberar o acesso. Tem-se como exemplo o *malware* Virobot, o qual funciona como um keylogger e ransonware.

- **Spam/Phishing:** Por meio de botnets é possível filtros de spam existentes. Dado que os IPs das máquinas infectadas não são geralmente conhecidos, não estão presentes em listas negras de spam. Além disso, os spams serão gerados de diversos servidores descentralizados, dificultando que sejam evitados. Atacantes podem, inclusive, utilizar os e-mails para realizar ataques de phishing, que ocorrem quando o atacante tenta obter informações sigilosas ao se disfarçar por uma entidade confiável.

- **Ataque de negação de serviço:** Conhecido como DDoS, é um ataque que tem como objetivo tornar serviços, em especial servidores web, indisponíveis. A partir de uma botnet, é possível coordenar os bots para realizarem múltiplas requisições a um determinado servidor, tornando-o indisponível. Um exemplo de botnet que realiza este tipo de ataque é a Cyclone.

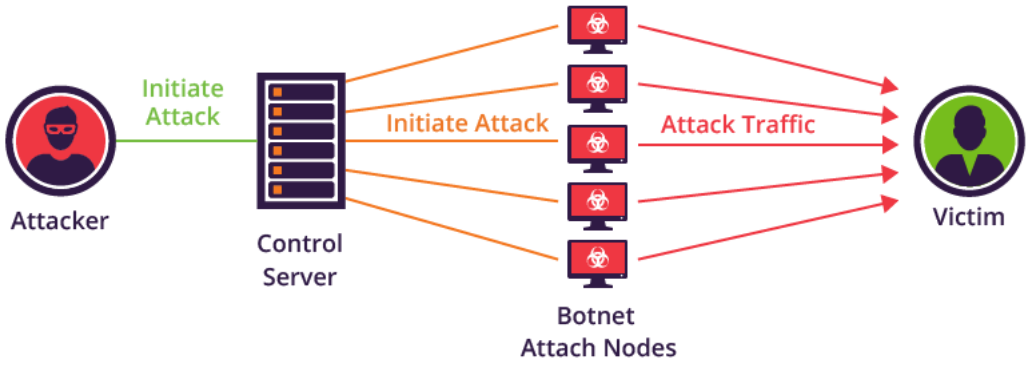


Figura 2 – Arquitetura de uma botnet DDoS
Fonte: Imperva [7]

2.6 Botnets notórias

A seguir são apresentadas algumas das botnets que ganharam notoriedade pela quantidade de bots infectados ou pelas dimensões do impacto financeiro gerado.

- **EarthLink Spammer**: Foi a primeira botnet a ganhar notoriedade, tendo enviado mais de 1,25 milhão de e-mails de phishing. Tinha como objetivo coletar informações sensíveis e transmitir vírus que enviassem informações remotamente.
- **Storm**: Uma das primeiras botnets P2P conhecidas. Estava envolvida em uma série de ataques criminosos que vão desde DDoS até roubo de identidades. Alguns servidores Storm foram desativados em 2008 e atualmente a rede é considerada parcialmente inativa.
- **Zeus**: É considerada a botnet mais ativa segundo a compahia de segurança Damballa, tendo infectado mais de 3,6 milhões de computadores nos Estados Unidos. Foi designada para acessar informações pessoais e é capaz de realizar ações maliciosas por meio de HTML injection em formulários.
- **Koobface**: Foi reportado que essa botnet infectou mais de 2,9 milhões de computadores nos Estados Unidos. Utiliza mensagens em redes sociais que incluem um link que supostamente seria para um vídeo. Ao clicar no link, é instalado um video codec com componente malicioso na máquina do usuário.
- **Mirai**: Já foi utilizada para ataques DDos massivos, deixando boa parte da Internet inacessível na costa leste dos EUA. No entanto, tornou-se notável por infectar milhares de dispositivos IoT inseguros.

2.7 Sistemas de detecção

Para se proteger contra botnets, ou qualquer outro tipo de *malware*, faz-se necessário um software capaz de identificar e alertar sobre a existência de códigos indesejáveis no sistema. Nesse contexto, os IDSs se tornam bastante importantes. Por meio deles é possível a coleta e o uso de diversos tipos de ataques conhecidos, de forma que se possa identificar tentativas de invasão.

Existem vários tipos de mecanismos que podem ser utilizados por um IDS para se identificar intrusos. Alguns deles se baseiam em padrões já identificados previamente sobre determinado tipo de ataque, enquanto outros seguem o princípio de que o comportamento de um intruso não é o mesmo de um usuário autorizado.

Conceitualmente, o IDS refere-se a um mecanismo que seja capaz de alertar sobre atividades intrusivas. Isso engloba qualquer ferramenta utilizada na descoberta de processos não autorizados de dispositivos de rede ou de computadores.

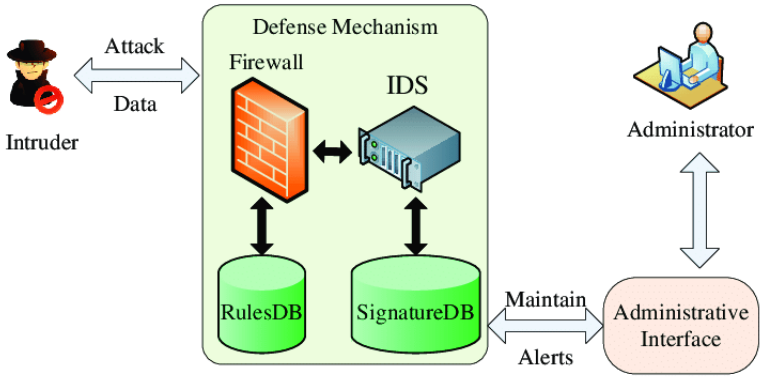


Figura 3 – Mecanismos de defesa
Fonte: Shi, Li e Feng (2018) [8]

2.7.1 Desafios na detecção de botnets

Dado o perigo que as botnets representam, vários sistemas de detecção já foram propostos para tentar identificar e mitigar os efeitos desse tipo de rede maliciosa. As principais dificuldades em se alcançar um IDS robusto e eficiente se devem a múltiplos fatores, entre eles:

- **Variedade de tipos de botnet:** Existem códigos de botnet para diversos fins, cada uma podendo apresentar comportamento totalmente distinto dos demais. Aspectos como os protocolos utilizados, a tarefa a ser executada pelas máquinas infectadas e a forma como a botnet se propaga pela rede são específicos de cada ataque, o que torna a detecção ainda mais díficil. As diferenças de topologias de botnets (C&C e P2P) também são impecilhos no processo de detecção.

- **Variações para um mesmo tipo de botnet:** Mesmo para dada botnet, os atacantes podem modificar o código de forma a mudar os padrões iniciais. Na tentativa de burlar sistemas de detecção, parâmetros como o tempo de comunicação e o tamanho dos pacotes enviados podem ser alterados. Para uma botnet de código aberto, é muito simples para um atacante inserir bytes adicionais ao corpo de dados (payload) para que o padrão de comunicação mude ou reconecte-se randomicamente para quebrar o fluxo estabelecido.

- **Falta de *datasets*:** Para o teste de um modelo proposto, é necessária a existência de *datasets* realísticos e com *malwares* atuais. Além da pouca quantidade de *datasets* públicos disponíveis, vários deles sofrem de problemas como dados corrompidos, pouca variedade de tráfego e inconsistências nos fluxos. Questões relacionadas à proteção de dados também são um limitante para a geração de *datasets* com tráfego benigno, visto que mesmo que a coleta de dados seja feita, a divulgação poderia violar a privacidade dos usuários.

As próximas seções apresentam algumas das abordagens propostas para o entendimento e detecção de botnets, bem como suas metodologias e limitações.

2.7.2 Honeypots

Honeypots são sistemas sem segurança usados como armadilhas para atrair atacantes e colher informações sobre o invasor. São montados fisicamente ou virtualmente separados da rede real, de forma que não possuam dados de valor e que os ataques sofridos não gerem impactos nas atividades normais da rede. É uma técnica efetiva para a coleta de informações sobre botnets, podendo aumentar o entendimento sobre o comportamento desse *malware* [9]. Através de honeypots é possível obter:

- Assinatura dos diferentes tipos de bots para uma base de *malwares*;

- Informações de servidores C&C;

- Falhas de segurança que possibilitam a entrada de botnets;

- Ferramentas usadas pelo atacante;

- Objetivo do atacante.

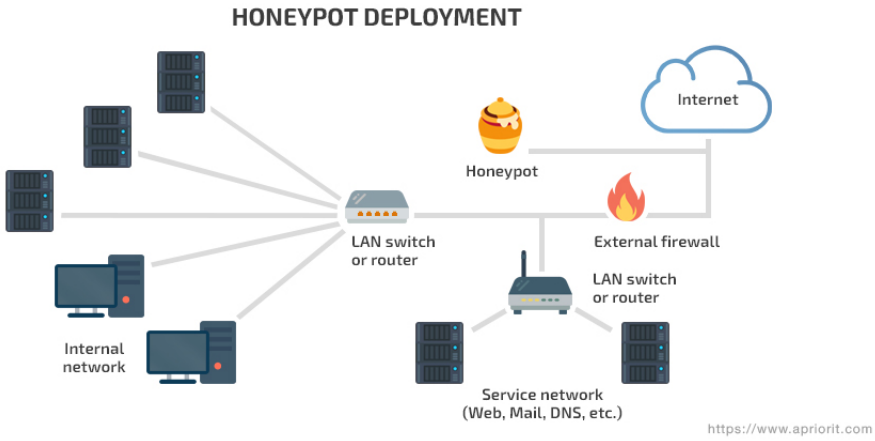


Figura 4 – Infrastrutura de rede com um honeypot
Fonte: Urasov (2019) [10]

Vários *datasets* publicamente disponíveis consistem de informações de honeypots. Esses *datasets*, no entanto, não são úteis para a coleta de tráfego benigno realístico, dado que são dedicadas ao tráfego de *malwares*. Apesar de serem bastante úteis para entendimento dos padrões de botnets, honeypots por si só não são úteis para identificação dos fluxos de bots. Para isso é necessário, por exemplo, uma técnica de análise baseada em assinatura ou baseada em anomalia.

2.8 Tipos de IDS

2.8.1 Baseado em assinatura

São sistemas que buscam determinados padrões já conhecidos (signatures) para a identificação de *malwares*. Verificam aspectos como a utilização de determinada porta, a existência de certo tipo de arquivo e os padrões de comunicação na rede de ataques comuns para alertar sobre a existência de códigos maliciosos. Essa abordagem é bastante precisa na detecção dos *malwares* presentes em seu banco de dados e não apresenta falsos positivos; no entanto, possui algumas graves limitações. Sendo somente capaz de identificar padrões pré-definidos, qualquer *malware* desconhecido também passaria imune a esse tipo de sistema. Mesmo *malwares* conhecidos com algumas simples alterações em seus parâmetros são capazes de passar despercebidos.

2.8.2 Baseado em anomalia

Sistemas por detecção de anomalias são abordagens que tentam detectar botnets baseados em parâmetros de rede fora do comum, como alta latência, grande volume de tráfego, comunicação por portas não usuais ou qualquer tipo de comportamente que difere muito do esperado. Podem ser divididos em:

a) Técnica baseada no hospedeiro: Também chamada de host-based technique, analisa as características e padrões dentro da máquina do usuário, buscando por anomalias referentes às funções do sistema operacional. Sistemas baseados nessa técnica costumam ser instalados de maneira individual, geralmente em máquinas corporativas ou servidores que armazenam dados de valor.

b) Técnica baseada na rede: Também chamada de network-based technique, tenta identificar botnets por anomalias nas características do tráfego na rede. São geralmente aplicados em interfaces de rede com fronteira de borda, de forma que todo o tráfego entre redes passe pelo IDS. Podem ser divididas em:

- Monitoramento ativo: Injetam pacotes de teste na rede, nos servidores ou aplicações, de forma a verificar as reações a esses pacotes. Produzem tráfego extra.

- Monitoramento passivo: Analisa os tráfegos de rede à medida que passam, sem inserir nenhum tipo de tráfego adicional. Pode ser feito em tempo real ou após o tráfego ter se encerrado. A maioria dos sistemas de detecção utilizam essa técnica.

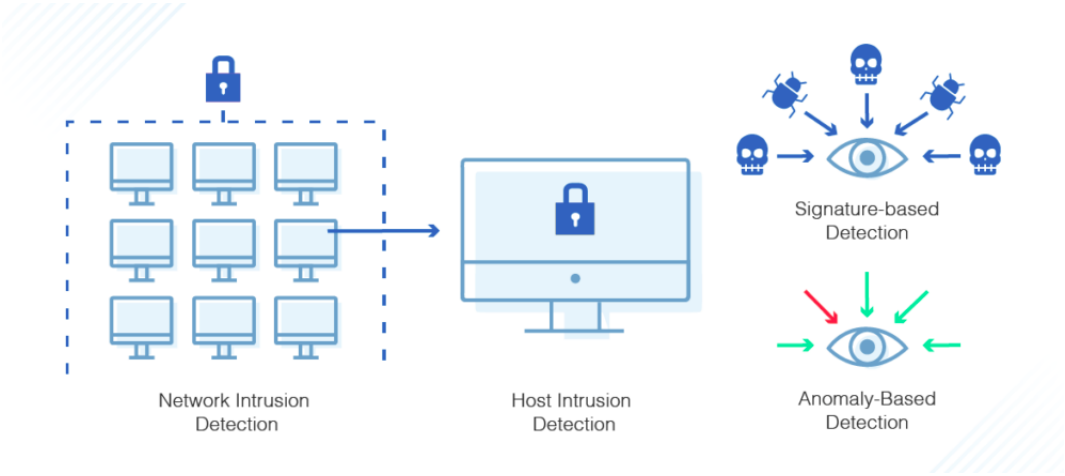


Figura 5 – Tipos de IDS
Fonte: SolarWinds Worldwide (2019) [11]

2.9 Aprendizado de máquina

O aprendizado de máquina (em inglês, *machine learning*) é um método de análise de dados que automatiza a construção de modelos analíticos. É um ramo da inteligência artificial baseado na ideia de que sistemas podem aprender com dados, identificar padrões e tomar decisões com o mínimo de intervenção humana. Dois dos métodos mais adotados de aprendizado de máquina são o aprendizado supervisionado e o aprendizado não supervisionado [12].

Algoritmos de aprendizado supervisionado são aplicados em situações em que a saída desejada é conhecida. O algoritmo recebe um conjunto de entradas juntamente com as correspondentes saídas corretas. Dessa forma, o modelo possui um gabarito para consultar e pode se auto ajustar de acordo com os erros da predição. São comumente empregados em aplicações nas quais dados históricos preveem eventos futuros prováveis.

Para analisar a efetividade desse tipo de técnica, é essencial que os dados disponíveis sejam divididos de forma aleatória em dois conjuntos: um conjunto de treino e um conjunto de testes. Se um modelo fosse testado com os dados de treino (os mesmos dados em que foi construído) seu resultado não poderia ser generalizado para o ambiente de produção, já que seria impossível saber o comportamento em dados nunca vistos. Por isso, existe o conjunto de testes. Normalmente, a maioria dos dados é usada para treinamento e uma parte menor dos dados é usada para teste. Porém, a organização e divisão dos dados pode ser feita de diferentes formas, que variam dependendo da qualidade e quantidade de dados coletados. Algumas dessas formas são as seguintes:

- **Validação por Hold-out**: Esta é a forma mais simples de separar os dados. Define-se um percentual para cada conjunto de dados (treino e teste). Geralmente este método é usado quando se tem uma grande quantidade de dados e sabe-se que as amostras usadas possuem significância estatística. O conjunto de treinos pode, adicionalmente, ser dividido em conjuntos de validação para ajuste do modelo durante a aprendizagem.

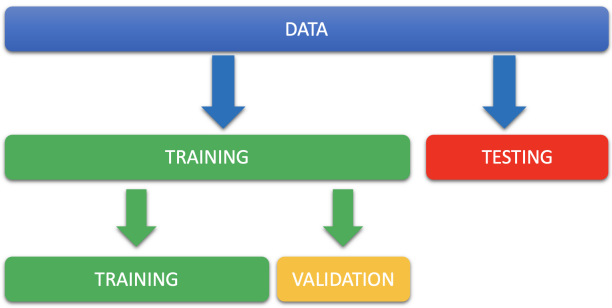


Figura 6 – Validação por Hold-out
Fonte: Sidhu (2019) [13]

- **Validação cruzada**: Existem vários tipos de validação cruzada, porém a mais comum é conhecida como validação cruzada de k-partições (*k-fold cross validation*). Neste método criamos um número k de amostras, sendo que cada amostra é deixada de lado enquanto o modelo treina com o restante delas. Para cada iteração, o modelo é testado com a amostra k que foi deixada de fora. No final, a qualidade do modelo é definida pelo desempenho médio de cada repetição. Valores comuns para k são 5 ou 10. Essa técnica costuma ser usada quando a quantidade de dados é pequena para se realizar a validação por hold-out.

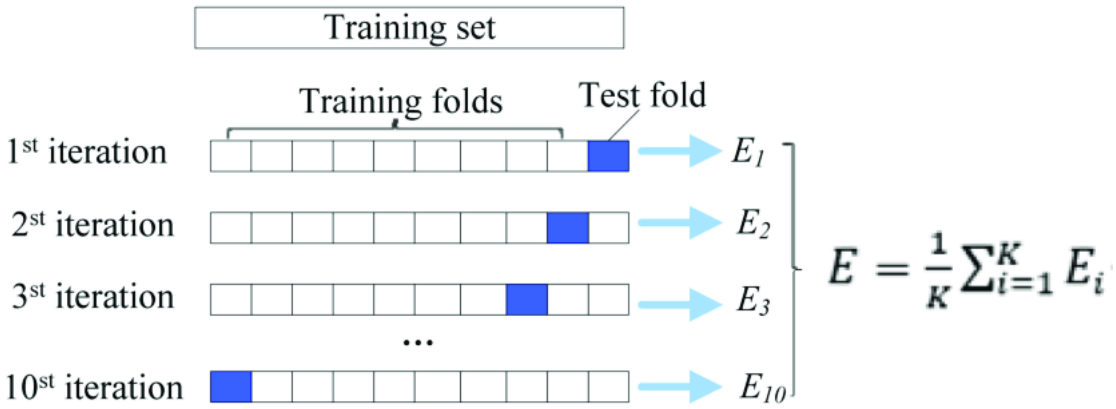


Figura 7 – Validação cruzada de k-partições
Fonte: Johar e Amer. (2019) [14]

Já nos algoritmos de aprendizagem não supervisionados, os dados fornecidos ao modelo não possuem rótulos históricos. Uma vez que o modelo não conhece a "resposta correta", deve explorar os dados e tentar organizá-los de acordo com alguma métrica. Essa abordagem é bastante útil quando se tem pouca ou nenhuma noção do resultado esperado ou quando se deseja identificar padrões não óbvios. Pode-se ser aplicada, por exemplo, na identificação de segmentos de clientes com atributos semelhantes para campanhas de marketing. Esses algoritmos são comumente utilizados para segmentar tópicos de texto, recomendar itens e identificar pontos discrepantes nos dados.

2.9.1 Redes neurais artificiais

Redes neurais artificiais (ANNs) são métodos computacionais baseados nas estruturas neurais de organismos inteligentes. São constituídos de camadas de nós (também chamados de neurônios ou perceptrons) conectados entre si por meio de arestas. Cada nó recebe um conjunto de dados como entrada e após algum processamento produz um conjunto de dados de saída. A Figura 8 ilustra a estrutura de uma rede neural.

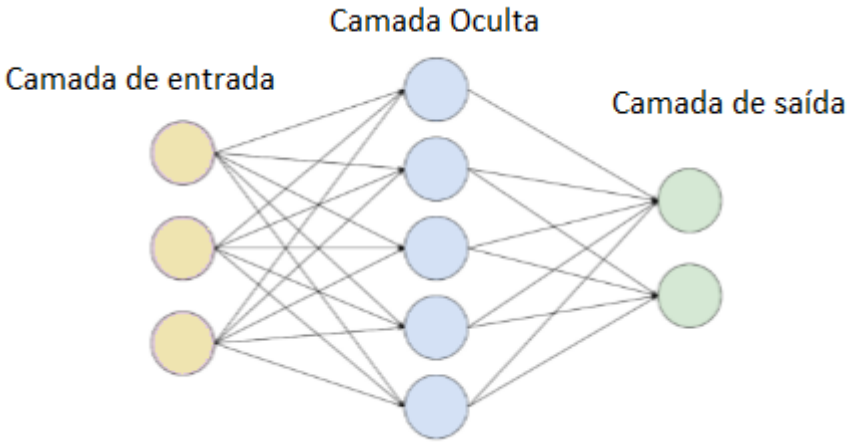


Figura 8 – Estrutura de uma Rede Neural
Fonte: Elaborada pelos autores

Observando-se a estrutura ilustrada na Figura 8, é possível perceber que uma rede neural é composta por 3 partes: camada de entrada, camada oculta e camada de saída. As arestas mostradas na imagem mostram como estão distribuídas as conexões entre os neurônios até a camada de saída, podendo cada uma destas arestas possuir pesos diferentes.

É por meio da camada de entrada que são introduzidos os dados que devem ser classificados. Os neurônios desta camada estão diretamente conectados a algum neurônio da camada oculta. Esta, por sua vez, tem a função de realizar os cálculos necessários para realizar as predições. Por meio de uma função de ativação, os sinais de entrada recebidas são convertidos em sinais de saída para as camadas seguintes. A camada oculta pode ser composta por uma ou mais camadas de neurônios. Por fim, tem-se a camada de saída, a qual é responsável por, de fato, produzir o resultado da classificação.

Durante o aprendizado de uma rede neural, baseado na diferença entre o valor predito e o valor correto esperado, uma função de custo quantifica o erro e o propaga de volta ao sistema para que os pesos das arestas sejam ajustados (algoritmo *backpropagation*). O objetivo desse processo, que ocorre diversas vezes durante o treinamento, é minimizar o erro. Quando todo o conjunto de dados de treino passa pela rede, diz-se que ocorreu um ciclo (epoch). Usualmente, várias epochs ocorrem antes que a rede esteja finalizada.

2.9.2 Árvores de decisão

A Árvore de Decisão (Decision Tree) é um algoritmo de aprendizado supervisionado utilizado em problemas de regressão e classificação. Este algoritmo é uma estrutura de árvore em forma de um fluxograma onde um nó interno é associado a uma *feature* e um ramo representa a regra de decisão, até chegar em uma folha - a qual representa o

resultado da predição. O nó mais alto de uma árvore é denominado nó raiz e representa a *feature* com maior importância na predição. A Figura 9 ilustra a estrutura de uma árvore de decisão.

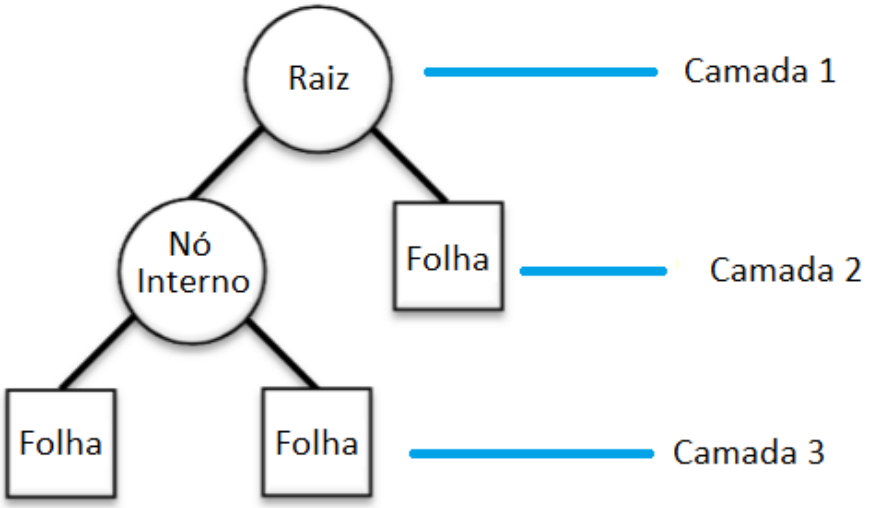


Figura 9 – Estrutura de uma Árvore de Decisão
Fonte: Elaborada pelos autores

O algoritmo principal para construção de uma árvore de decisão é chamado de ID3. Criado por Ross Quinlan, esse algoritmo foi baseado em sistemas de interferência. O ID3 separa um conjunto de amostras em conjuntos menores, de modo que estes possuam uma única classe, ou seja, pertençam a um único tipo. O ID3 utiliza tanto a entropia quanto o ganho de informação para realizar a contrução da árvore.

- **Entropia**: Similarmente ao conceito termodinâmico, a entropia pode ser entendida como a variação do tipo dos dados. Um *dataset* bem balanceado, ou seja, que possua aproximadamente o mesmo número de amostras para cada classe possui uma entropia alta. Já conjuntos de dados que possuam uma grande predominância de uma classe tem uma entropia baixa. A entropia de um conjunto de dados pode ser calculada da seguinte forma:

$$E = -\sum_{i}^{C} p_i log_2 p_i \tag{2.1}$$

  Onde E representa a entropia do conjunto de dados e p_i a probabilidade de escolher aleatoriamente um elemento da classe i.

- **Ganho de Informação**: O ganho de informação pode ser entendido como uma métrica baseada na diminuição da entropia após a divisão de um conjunto de dados

em um atributo. Esta medida é calculada para se definir em que camada da árvore as *features* ficarão. As *features* que tiverem ganho de informação mais relevante ficarão na parte superior da árvore. O ganho de informação pode ser calculado como mostrado a seguir:

$$Ganho(S, A) = Entropia(S) - \sum_{v \in A} p(A_v) * Entropia(A_v) \tag{2.2}$$

Em que S representa o conjunto todo e A representa o atributo em questão.

Dessa forma, com a árvore de decisão construída, cada amostra que deverá ser classificada deverá passar por seus nós até chegar em uma folha e ser predita.

2.9.3 Mapas auto-organizáveis

Mapas auto-organizáveis ou self-organizing maps (SOMs) são um tipo de ANN que opera sob aprendizado não supervisionado. Possuem a capacidade de organizar dimensionalmente dados complexos em grupos, de acordo com suas relações. São usados para gerar uma representação discretizada e de poucas dimensões (usualmente duas dimensões) do conjunto de entrada fornecido. Portanto, é um método de redução de dimensionalidade.

Difere de outras redes neurais por usar aprendizado competitivo ao invés de aprendizado por correção de erro e *backpropagation*. As etapas de mapeamento do SOM começam com a inicialização dos vetores de peso. A partir daí, um vetor de amostra é selecionado aleatoriamente e o neurônio cujo vetor peso mais se aproximar do vetor de amostra é ativado e considerado o neurônio "vencedor". Essa etapa é conhecida como etapa de competição. Em sequência, o neurônio vencedor ativa os neurônios dentro de sua vizinhança. A definição da vizinhança será dada por dois parâmetros: a topologia da rede e o raio de vizinhança. Na última etapa, conhecida como etapa de adaptação, o neurônio vencedor e sua vizinhança têm seus pesos ajustados e passam a armazenar as características dos padrões de entrada que os ativaram [15]. Ao final dessas etapas, será formado um mapa de saída agrupando os dados de entrada segundo características similares.

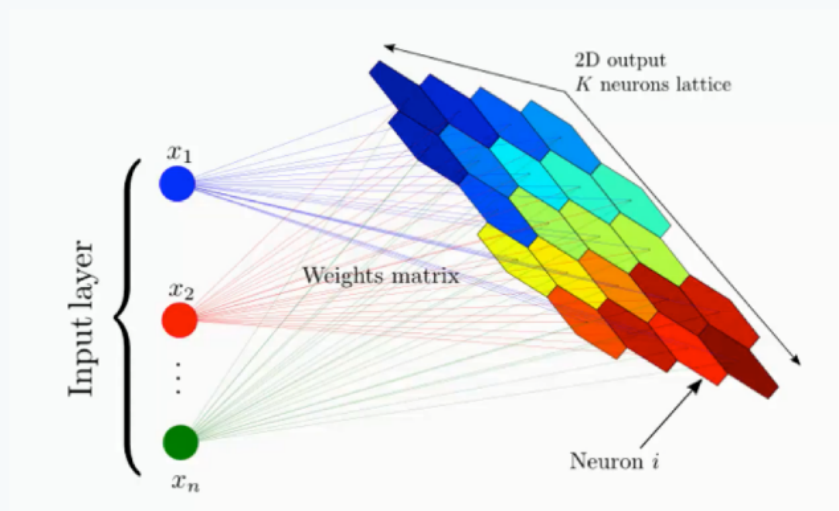


Figura 10 – Self-organizing map
Fonte: SuperDataScience Team (2018) [16]

2.9.4 Métricas de avaliação

Após o processo de criação de um modelo de aprendizado de máquina, é necessário medir sua qualidade através de alguma métrica. Tão importante quanto saber escolher um bom modelo é saber escolher a métrica correta para decidir qual é a melhor suas possíveis variações. Para isso, algumas funções matemáticas são utilizadas para avaliar a capacidade de predição do modelo. Ao escolher uma métrica, é importante levar em consideração fatores como a proporção de elementos em cada classe no *dataset* e o objetivo da previsão (por exemplo, dando prioridade para deteccção de bots). Supondo que as duas possíveis classes que podem ser preditas por um modelo sejam "Bot"(positivo) e "Benigno"(negativo), algumas definições importantes para o cálculo dessas métricas são:

- **Verdadeiro Positivo (VP)**: Número de entradas maliciosas corretamente classificadas como "Bot".
- **Verdadeiro Negativo (VN)**: Número de entradas benignas corretamente classificadas como "Benigno".
- **Falso Positivo (FP)**: Número de entradas benignas incorretamente classificadas como "Bot".
- **Falso Negativo (FN)**: Número de entradas maliciosas incorretamente classificadas como "Benigno".

Essas métricas podem ser apresentadas em um arranjo (layout) de tabela específico, chamado de matriz de confusão. Para as classes definidas anteriormente, a matriz de confusão seria a seguinte:

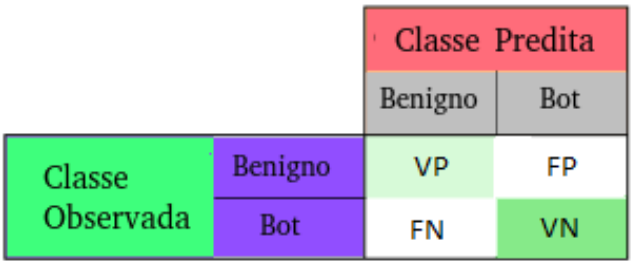


Figura 11 – Matriz de confusão de referência
Fonte: Elaborada pelos autores

Acurácia:

Estima a proporção de reconhecimento correto em relação a todos os registros de todo o conjunto de teste. Quanto maior a precisão, melhor é o modelo utilizado (Acurácia $\in$ [0, 1]). É uma boa medida para conjuntos de dados que contém classes balanceadas [17]. É definida como:

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.3}$$

Precisão:

Estima a proporção de registros de bots corretamente classificados em relação ao número total de registros classificados como bot. Quanto maior a precisão, melhor é o modelo utilizado (Precisão $\in$ [0, 1]). É definida como:

$$Precisão = \frac{VP}{VP + FP} \tag{2.4}$$

Taxa de Verdadeiros Positivos (TVP):

Também chamada de *recall*, estima a proporção de reconhecimento correto para registros de bots em relação a todos os bots existentes no conjunto de testes. Quanto maior a TVP, melhor o modelo utilizado (TVP $\in$ [0, 1]) .

$$TVP = \frac{VP}{VP + FN} \tag{2.5}$$

Taxa de Falso Negativo (TFN):

Estima a proporção de registros benignos incorretamente classificados em relação ao número total de registros benignos no conjunto de testes. Quanto menor a TFN, melhor

o modelo utilizado (TFN $\in$ [0, 1]).

$$TFN = \frac{FP}{FP + VN} \tag{2.6}$$

F1-Score:

É a média harmônica entra a precisão e o recall. É uma boa métrica quando a distribuição de classes é desproporcional e o modelo não emite probabilidades. Quanto maior o F1-Score, melhor o modelo utilizado (F1-Score $\in$ [0, 1]).

$$F1 - Score = 2 * \frac{Precisão * Recall}{Precisão + Recall} \tag{2.7}$$

Área sob a curva Característica de Operação do Receptor

Comumente chamada de AUC (*Area under the curve*), calcula-se por meio da área sob uma curva formada pelo gráfico entre a taxa VP e a taxa de FP. É uma métrica interessante para tarefas com classes desproporcionais. Uma das vantagens em relação ao F1-Score, é que ela mede o desempenho do modelo em vários pontos de corte, não necessariamente atribuindo exemplos com probabilidade maior que 50% para a classe positiva, e menor, para a classe negativa.

Em sistemas que se interessam apenas pela classe, e não pela probabilidade, ela pode ser utilizada para definir o melhor ponto de corte para atribuir uma ou outra classe a um exemplo. Este ponto de corte normalmente é o ponto que se localiza mais à esquerda, e para o alto, no gráfico [18].

Na concepção de um modelo de aprendizado de máquina, busca-se que este não seja nem muito flexível e se comporte como um bom preditor somente para os dados que lhe foram fornecidos no treinamento (o que caracteriza *overfitting*), nem tão genérico a ponto de não modelar fielmente os dados (*underfitting*).

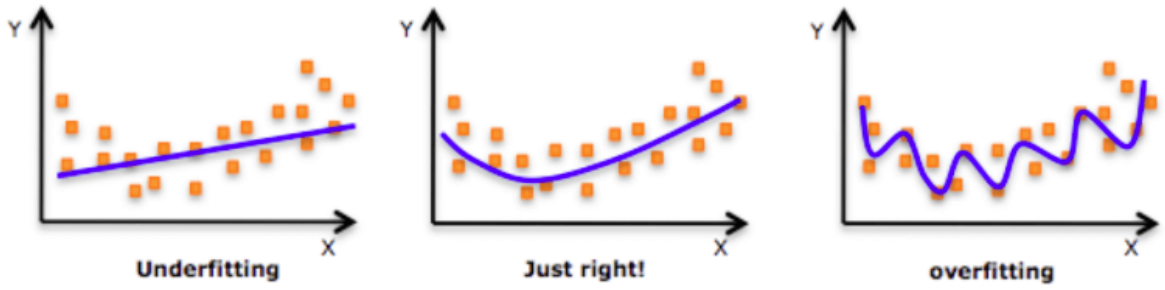


Figura 12 – Representação gráfica de *overfitting* e *underfitting*
Fonte: Amazon [19]

3 Abordagem Utilizada

A abordagem utilizada no trabalho pode ser sintetizada pelas etapas presentes na Figura 13. O primeiro passo seria uma captura de rede realizada por meio de simulações em ambientes reais ou virtuais. A partir dos fluxos de rede coletados, informações (*features*) devem ser extraídas para uso posterior.

Um fluxo de rede pode ser definido como um resumo da conexão entre dois hospedeiros. É estabelecido com base na combinação do protocolo, dos IPs de origem e destino e das portas de origem e destino. Esses fluxos podem prover informações valiosas quanto às atividades da rede e de quem a utiliza. Existem diversas formas de extraí-los, sendo uma das maneiras mais populares o uso de ferramentas de análise de tráfego [20].

O segundo passo é selecionadar as melhores *features* para o treinamento de um modelo de classificação que possa diferenciar máquinas legítimas de máquinas que fazem parte de uma botnet. O terceiro passo é a construção do modelo de classificação usando o conjunto de *features* selecionadas. Esse modelo deve ser avaliado por meio de uma técnica de validação, como por exemplo, validação cruzada de acordo com métricas que indicam a qualidade da predição, como a acurácia ou o recall.

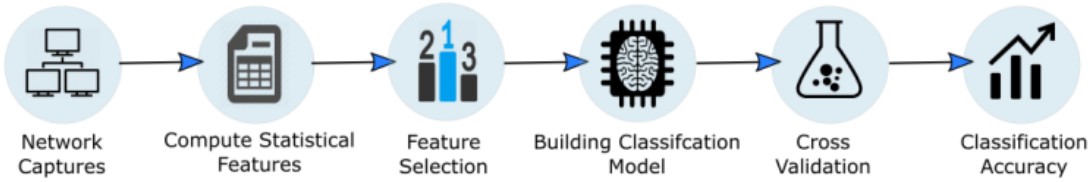


Figura 13 – Metodologia utilizada
Fonte: Pektaş e Acarman (2017) [20]

Uma vez que o intuito desse trabalho não é realizar a captura do tráfego de rede de botnets, esses dados foram coletados de simulações previamente realizadas e disponíveis publicamente. Para isso, bancos de dados (*datasets*) com os fluxos desejados devem ser escolhidos. As demais etapas foram realizadas e são detalhadas no decorrer do trabalho.

3.1 Escolha dos *datasets*

A escolha do(s) *dataset(s)* a ser(em) utilizados é um fator decisivo para a eficiência do sistema de detecção. Além da quantidade de *datasets* disponíveis ser limitada, muitos dos *datasets* já montados não são disponíveis publicamente devido a questões de segurança e privacidade. Além disso, a construção de um conjunto de dados restrige-se a um certo escopo de ataques e tipos de tráfego. O que se busca é um *dataset* que seja o mais completo e menos limitado possível.

De acordo com Sperotto et al., 2009 [21], um bom *dataset* deve conter tráfego de botnet e tráfego benigno devidamente identificados e o balanço do *dataset* deve ser similar a uma rede real (usualmente o percentual de tráfego de bots é pequeno).

Os principais aspectos desejáveis são os seguintes:

- Variedade de tráfego;
- Tráfego que se assemelhe ao de uma rede real;
- Variedade de ataques (botnets distintas);
- Uso de botnets reais e não simuladas;
- Diferenciação entre tráfegos benignos e malignos;
- Tamanho significativamente grande.

3.2 *Datasets* disponíveis

Esta seção se limita a detalhar alguns dos mais relevantes e completos *datasets* públicos. Optamos por utilizá-los como base para o sistema de detecção proposto por estarem de acordo com as características desejáveis descritas acima e terem sido construídos em um período reletivamente recente, o que implica ataques também atuais.

3.2.1 CTU-13:

Esse *dataset* foi capturado na Czech Technical University, em Prague na República Tcheca, no ano de 2011, com o objetivo de prover uma larga captura de tráfego real de botnet misturado a tráfego benigno. Consiste em 13 cenários com diferentes tipos de botnets, com protocolos e finalidades distintas. É um dos maiores *datasets* disponíveis e foi usado em trabalhos como o de Grill et al. [22] para avaliar o Snort-IDS na detecção de botnets e o de Wang et al. [23] para análise da similaridade entre os fluxos de bots.

Cada cenário foi capturado em um arquivo de formato pcap que contém os dois tipos de tráfegos citados acima. Esses arquivos foram processados para extrair informações

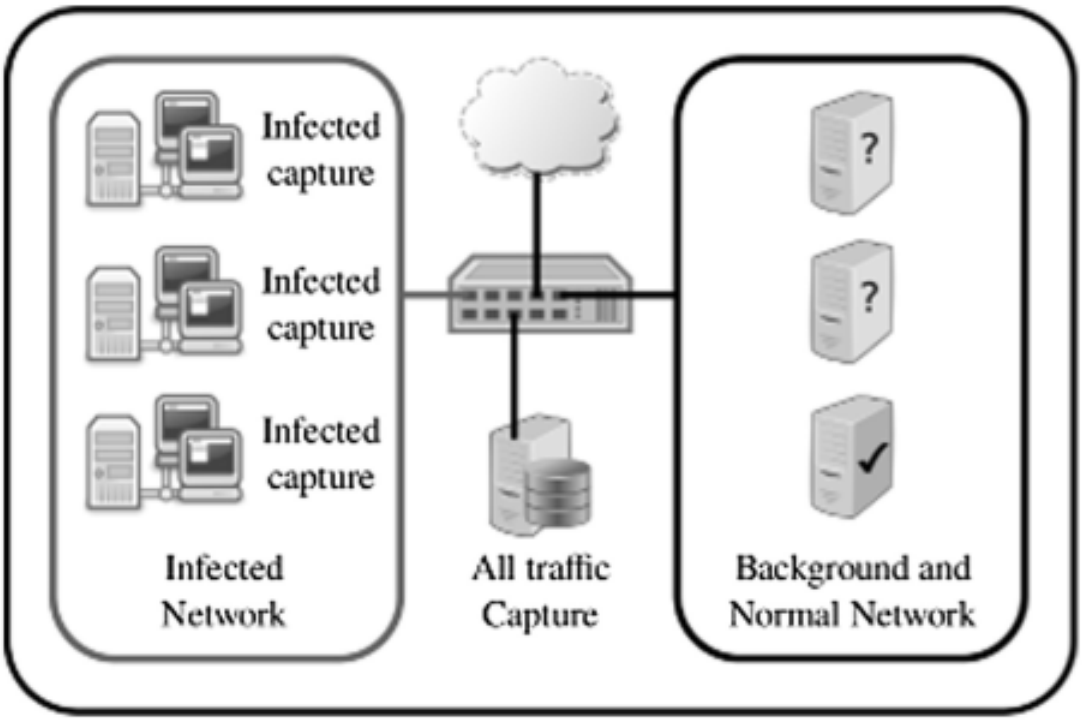


Figura 14 – Topologia simplificada para construção do *dataset* CTU-13
Fonte: Stratosphere Lab (2014) [24]

características da conexão. Contêm fluxos bidirecionais, o que diferencia cliente e servidor [25].

Um cénario no CTU-13 *dataset* pode ser definido com uma infecção particular de uma máquina virtual que executa determinado *malware*. Cada uma dessas máquinas foi colocada em modo bridge na rede da universidade. A Figura 14 demonstra uma simplificação da topologia montada.

O período de coleta entre cénarios é significativamente diferente. Múltiplos tipos de bots são encontrados, sendo a maior parte dos cenários composta por somente um bot (cenários 1 a 8 e 13), enquanto que os cenários de 9 a 12 contêm múltiplos bots simultâneos. O percentual de fluxos de bot é negligenciável (menor que 2%) se comparado ao fluxo total nos cenários de apenas um bot. Para os outros cenários, esse percentual aumenta (entre 6% e 8%) [26].

A duração de cada cenário, a quantidade de fluxos de rede, o tamanho do pcap capturado, a quantidade de bots utilizados e a porcentagem do tráfego de bots são mostrados na tabela 1.

Dataset	Duração (h)	Fluxos	Tamanho (GB)	Bot	Nª de Bots	Fluxos de Bot
1	6,15	2.824.637	52	Neris	1	39.933 (1,41%)
2	4,21	1.808.123	60	Neris	1	18.339 (1,04%)
3	68,85	4.710.639	121	Rbot	1	26.759 (0,56%)
4	4,21	1.121.077	53	Rbot	1	1.719 (0,15%)
5	11,63	129.833	37,6	Virut	1	695 (0,53%)
6	2,18	558.920	30	Menti	1	4.431 (0,79%)
7	0,38	114.078	5,8	Sogou	1	37 (0,03%)
8	19,5	2.954.231	123	Murlo	1	5.052 (0,17%)
9	5,18	2.753.885	94	Neris	10	179.880 (6,5%)
10	4,75	1.309.792	73	Rbot	10	106.315 (8,11%)
11	0,26	107.252	5,2	Rbot	3	8.161 (7,6%)
12	1,21	325.472	8,3	NSIS.ay	3	2.143 (0,65%)
13	16,36	1.925.150	24	Virut	1	38.791 (2,01%)

Tabela 1 – Bots por cenário

Foram disponibilizados arquivos de extensão binetflow, que contém dados como endereços de origem e destino, duração do fluxo, protocolo utilizado e tamanho em bytes das informações trocadas. Não estão disponíveis algumas *features* mais detalhadas sobre o fluxo; no entanto, essas *features* podem ser extraídas utilizando ferramentas como o CICFlowMeter ou o Netflow Analyzer.

3.2.2 CIC 2018:

Este *dataset* foi criado pelo Canadian Institute for Cybersecurity (CIC) com o intuito de auxiliar a análise, teste e avaliação de sistemas de detecção de intrusão, com base em análise de anomalias em uma rede. Foram construídos diversos cenários e, capturando-se os pacotes, realizaram-se 7 tipos de ataques: Brute-force, Heartbleed, Botnet, DoS, DDoS, Ataques Web e Ataques de Intrusão. No cenário criado para a realização dos ataques, foram utilizadas 50 máquinas para realização de diferentes ataques a uma organização que continha 420 hospedeiros e 30 servidores. A Figura 15 ilustra a topologia utilizada para a construção do *dataset*.

Como este trabalho é focado na detecção de botnets, apenas será explicado como foi gerado o *dataset* para esse tipo de ataque. O *dataset* em questão foi construído em apenas um dia, tendo utilizado um computador com Kali Linux como atacante e como vítima diversos sistemas operacionais, como Windows Vista, 7, 8.1 e 10.

Além disso, o ataque foi feito por meio da botnet Zeus, a qual já foi empregada para infectar milhões de computadores ao redor do mundo. Apesar de ser útil para executar tarefas maliciosas, este *malware* é muito utilizado para obter informações bancárias através de keylogging e captura de formulários. Também possui a capacidade de baixar e instalar programas sem a permissão do usuário, motivo pelo qual também é serve para a instalação

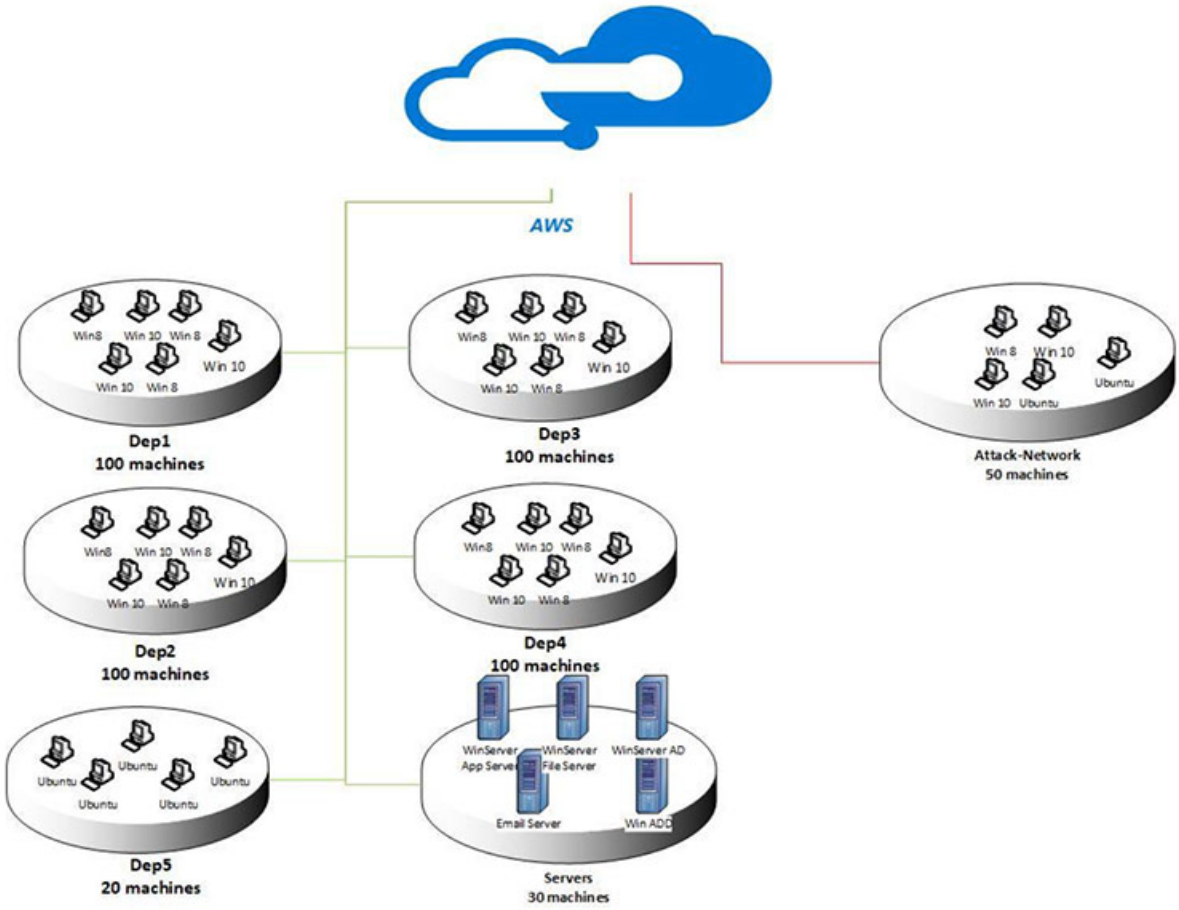


Figura 15 – Cenário utilizado para construção do *dataset* CIC 2018
Fonte: Canadian Institute for Cybersecurity Website [27]

de ransomwares, como o Crypto-Locker. Ademais, a Zeus pode se espalhar facilmente por uma rede, principalmente por meio de downloads e de esquemas de phishing.

Entretanto, para a construção do *dataset* de botnet não foi utilizada somente a Zeus nas simulações. Como um complemento, usou-se a Ares, uma botnet de código aberto com a capacidade de realizar keylogging, capturas de tela das vítimas, execução de comandos shell remotos, download e upload de arquivos. Dessa forma, neste cenário de ataque, os computadores das vítimas foram infectados tanto com a botnet Zeus quanto com a Ares, ocorrendo captura de tela a cada 400 segundos.

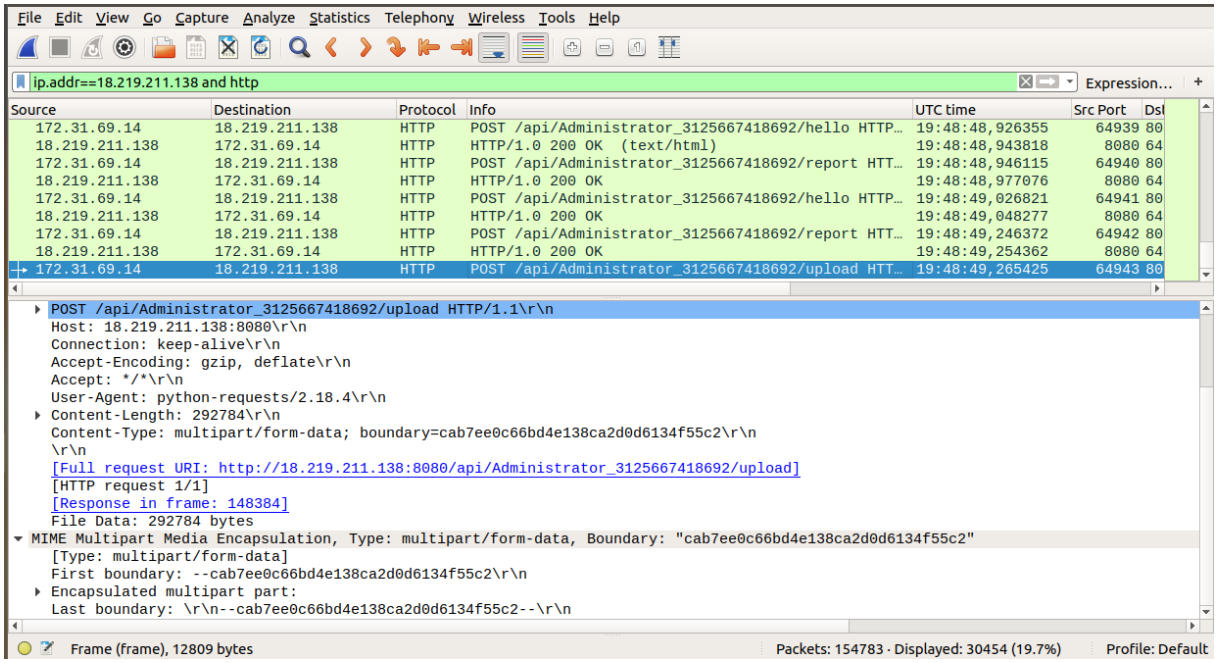


Figura 16 – Pacote de envio de screenshot visualizado no Wireshark
Fonte: Captura de tela do Wireshark

Após terminada a simulação, foi utilizada a ferramenta CICFlowMeter para a extração das *features* e construção do *dataset* em formato csv. Este arquivo possui mais de 80 colunas, cada uma contendo características referentes ao fluxo gerado - como por exemplo a duração do fluxo, tamanho máximo de um pacote recebido, tempo total entre dois pacotes enviados. Esse *dataset* foi utilizado em trabalhos como o de [17] para a construção de um IDS baseado em deep learning, e em [28] para um IDS por rede neural.

3.2.3 CIC 2017:

Assim como o *dataset* CIC 2018, foi montado pelo CIC e se encontra disponível na forma de arquivos pcap ou de arquivos csv com os fluxos rotulados e *features* extraídas pelo CICFlowMeter. O período de captura foi de 5 dias, tendo sido iniciado na manhã do dia 03/07/2017 e encerrado às 5 da tarde do dia 07/07/2017. As capturas referentes a botnets ocorreram entre 10h e 11h da manhã do dia 7.

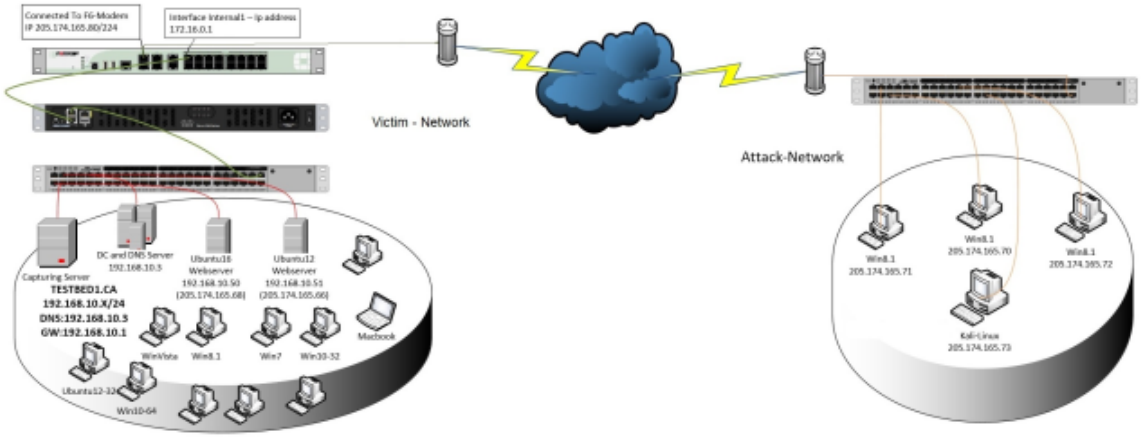


Figura 17 – Cenário utilizado para construção do *dataset* CIC 2017
Fonte: Sharafaldin, Lashkari e Ghorbani (2018) [29]

A metodologia utilizada para construção desse *dataset* é semelhante à descrita para o *dataset* CIC 2018. No entanto, ao invés da utilização da botnet Zeus como principal, foi utilizada somente a botnet Ares.

Attack	Tools	Duration	Attacker	Victim
Botnet attack	• Ares (developed by Python): remote shell, file upload/download, capturing • screenshots and key logging	One day	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)

Figura 18 – Informações referentes à simulação de botnet no *dataset* CIC 2017
Fonte: Sharafaldin, Lashkari, Hakak e Ghorbani (2019) [30]

4 Análise Baseada Em Fluxos De Rede

Com o crescimento do uso de botnets e de sua periculosidade, muitos pesquisadores e profissionais de segurança construíram diferentes técnicas para detectar estes *malwares*. Pode-se citar como exemplo os métodos de detecção por comportamento e por assinatura. Estas diferentes técnicas são baseadas em diferentes abordagens, no entanto, metodologias baseadas em aprendizado de máquina podem ser aplicadas em ambos os casos.

Uma abordagem de detecção por meio de assinaturas exige um grande conhecimento sobre as características que um bot pode possuir. O método é utilizado para identificar características específicas, como um protocolo em particular, funcionando como antivírus comuns que conhecem assinaturas específicas de *malwares*, conseguindo identificar apenas o que está em seu banco de dados - uma simples mudança pode tornar o *malware* indetectável. Por isso, a abordagem somente é eficaz contra botnets já conhecidas, tornando-se ineficiente contra bots desconhecidos e mais suscetível a ser burlada [31].

Os sistemas baseados no comportamento do bot são construídos com base nos "hábitos" de uma botnet, como por exemplo a frequência que um bot se comunica com o C&C. Quanto mais genérico forem estes sistemas, maiores serão as chances de identificar bots desconhecidos, mas se forem demasiadamente genéricos, o número de falsos positivos pode crescer bastante [28].

Já as abordagens de detecção baseadas em aprendizado de máquina têm se mostrado muito efetivas para deteção de botnets [32]. Essa eficácia se dá devido a sua capacidade de identificar tráfego de um bot em meio a um tráfego normal. Isso é um desafio para outros tipos de abordagem, já que os protocolos que as botnets passaram a usar são os mesmos de uma comunicação benigna, como o HTTP. Entretanto, este método de detecção requer uma quantidade considerável de exemplos de treino e *features* bem definidas para uma classificação efetiva.

Dessa forma, pelas vantagens descritas anteriormente, este trabalho irá focar em métodos de detecção de botnets utilizando aprendizado de máquina.

4.1 Extração de *features* de arquivos pcap

Dado que as simulações são disponibilizadas em arquivos do tipo pcap, para extrair informações úteis dos pacotes da rede faz-se necessário o uso de alguma ferramenta de análise de tráfego. Em ambos os *datasets* do CIC, juntamente com os pcaps, foram disponibilizados arquivos csv já prontos com diversas *features* de fluxos. Esses arquivos foram

gerados a partir de uma ferramenta chamada CicFlowMeter. Pela praticidade e qualidade desse software, optou-se por utilizá-lo para extrair os mesmos dados dos *datasets* da CTU.

4.1.1 CicFlowMeter

É um gerador de fluxos de rede escrito em Java e de código aberto que oferece flexibilidade em termos de escolher as *features* para análise ou cálculo, sendo possível a adição de novas *features* ou ajuste das existentes. Gera fluxos bidirecionais, em que o primeiro pacote determina o sentidos direto (fonte para destino) e reverso (destino para fonte). Conexões por meio do protocolo TCP são usualmente consideradas encerradas pelo envio de pacotes FIN, enquanto conexões UDP são terminadas por *timeout* de fluxo. O *timeout* pode ser arbritariamente escolhido para ambos os protocolos [33]. Permite a análise de fluxos em arquivos já gerados (offline) ou em tempo real a partir da interface da máquina (online). É capaz de extrair mais de 80 *features* de arquivos pcap. A imagem a seguir mostra a interface do CicFlowMeter, bem como sua utilização para extração de *features* do *dataset* 42 da CTU.

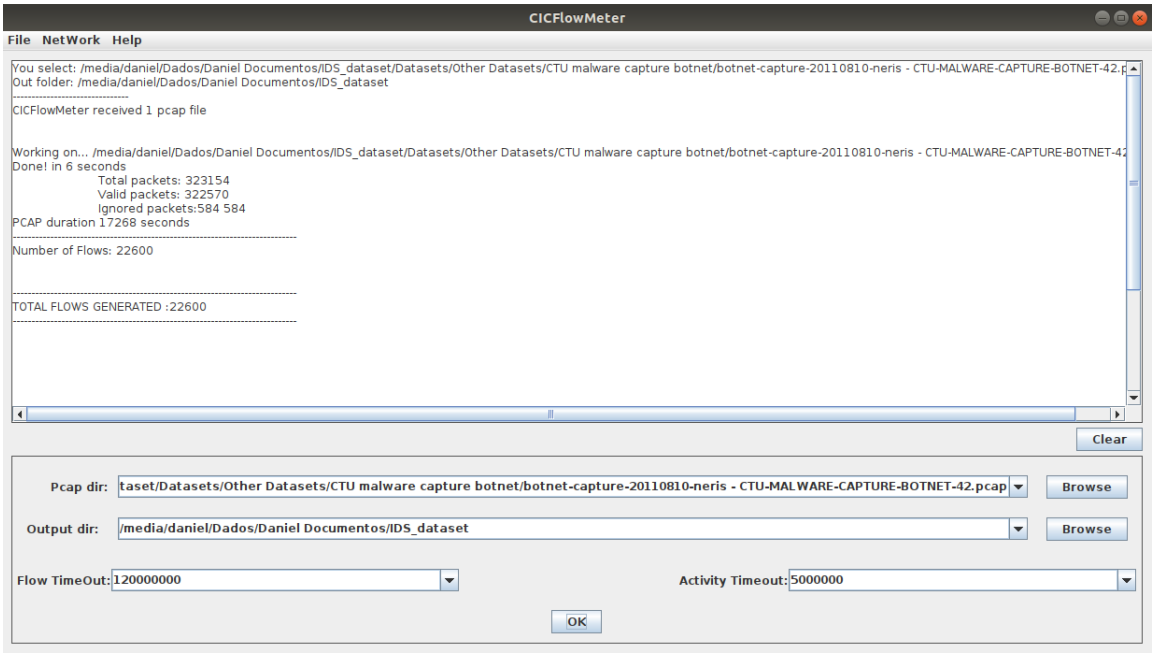


Figura 19 – Interface do CicFlowMeter
Fonte: Elaborada pelos autores

4.2 Seleção de *features*

Dada a escolha por algoritmos de aprendizado de máquina, faz-se necessário a escolha das *features* a serem utilizadas para o treinamento do modelo. A princípio, foram descartadas quaisquer *features* obviamente irrelevantes para o projeto, como a marca

temporal (timestamp) ou aquelas que não sofriam nenhuma variação no *dataset* inteiro, uma vez que não seriam úteis para diferenciação de fluxos.

Em seguida, foram descartadas *features* que, apesar de terem sido utilizadas em outros sistemas de identificação, tornam a detecção bastante limitada, como por exemplo o protocolo. As primeiras botnets utilizavam IRC, um protocolo projetado para internet chat, para comunicação com as máquinas infectadas. Como resultado, diversos métodos de detecção passaram a explorar esse aspecto [34]. No entanto, as bots mais recentes utilizam canais de comunicação mais robustos com protocolos como HTTP e P2P. Com intuito de tornar a detecção independente do protocolo utilizado, essa *feature* também foi ignorada. Da mesma forma para as portas de destino e origem e flags TCP.

Iniciou-se então um processo de identificação das principais *features* que pudessem apresentar um indicativo de bots na rede, tanto pelas características intrínsecas de botnets quanto pelo sucesso no uso de tais *features* em trabalhos relacionados. Algumas das selecionadas para análise foram as seguintes:

Duração do fluxo: É um dos parâmetros que mais pode variar para diferentes botnets; no entanto, muitas botnets costumam manter sessões de comunicação longas. Essa *feature* já foi largamente utilizada em sistemas propostos [35]. Ao observar o *dataset* do CIC 2018, visualmente essa *feature* aparentava ter valores pequenos para fluxos benignos e valores grandes para fluxos de bot.

Tamanho em bytes dos fluxos: A quantidade de bytes trocados tanto nos fluxos de cliente para servidor, como no sentido contrário, tendem a representar padrões de comunicação similares. A argumentação por trás do uso desse tipo de *feature* é de que o tráfego gerado por bots tende a ser mais uniforme do que o tráfego gerado por usuários legítimos [36].

Tamanho em bytes dos pacotes: Semelhante à *feature* anterior, no entanto, fazendo a análise dos pacotes individualmente.

Quantidade de pacotes por fluxo: Baseado na suposição de que bots tentam manter suas conexões ativas, usualmente tendem a enviar uma grande quantidade de pacotes.

Tempo médio entre o envio de 2 pacotes: Podem ser úteis na identificação de bots que apresentam uma periodicidade no envio de mensagens. Máquinas infectadas podem também fazer requisições aos servidores de C&C para recuperar comandos do botmaster em intervalos regulares.

Tamanho do primeiro pacote: É obtido imediatamente quando o fluxo se inicia e é carregado nas janelas de tempo futuras. Em muitas botnets, a troca inicial de pacotes ao entrar na rede tende a ser única com comportamente muito bem definido.

Tendo como base as *features* acima descritas, iniciou-se o processo de treinamento do sistema de aprendizado de máquina. Esse processo inicial não tinha como objetivo uma alta taxa de detecção de bots, mas sim a análise e escolha de hiperparâmetros para o modelo escolhido.

4.3 Treinamento do modelo

Inicialmente, foi escolhido o algoritmo máquina de vetores de suporte (SVM) para o treinamento do sistema pelo fato de ser eficiente em espaços de alta dimensão e ser relativamente eficiente em memória. Como cada amostra do *dataset* separada para treinamento possui n *features*, cada amostra pode ser vista como um ponto em um espaço de dimensão n. Dessa forma, o classificador irá procurar um conjunto de hiperplanos que melhor separará os pontos com características de botnet de pontos característicos de uma comunicação normal. Entretanto, o altíssimo tempo de convergência tornou o modelo inviável, sendo necessária a escolha de um novo método.

Foi escolhido, então, o uso de redes neurais artificiais para a detecção de fluxos de botnet por meio da análise do tráfego dos *datasets*. A arquitetura utilizada foi de uma *feedforward neural network* (FNN), que é um tipo de ANN que não forma ciclos. Apesar de se fazer necessária uma quantidade de dados maior para treinamento e ser computacionalmente mais caro se comparado a outros modelos de aprendizado de máquina, o algoritmo possui características vantajosas para a resolução do problema, como a capacidade de aprender e modelar relacionamentos complexos e não lineares.

4.3.1 Keras

Foi escolhido o Keras para a construção da rede neural por ser amplamente utilizado em pesquisas que envolvem Deep Learning e pela simplicidade e facilidade de uso. O Keras é uma interface de programação de aplicações (API) escrita em Python a qual é capaz de operar em cima de bibliotecas robustas, como o TensorFlow ou Theano.

4.3.2 Hiperparâmetros

Um dos pontos cruciais para o treinamento de redes neurais do tipo perceptron multicamadas (MLP) com treinamento por backpropagation diz respeito à definição de seus parâmetros. Pequenas diferenças nos parâmetros podem levar a resultados e generalizações completamente distintas. Não existe uma fórmula para se definir quais parâmetros são adequados para cada problema. O máximo que se pode encontrar são sugestões e recomendações baseadas em experiências, que podem levar a bons resultados para diversos cenários.

Dado que não se sabe, a priori, quais são os hiperparâmetros ideais a serem usados, foi definido um conjunto inicial e sendo feitos ajustes com base em testes e na resposta da rede para cada alteração. A seguir são apresentados os hiperparâmetros a serem decididos, juntamente com as recomendações de acordo com a literatura.

Número de Camadas Ocultas: Testes empíricos com a rede neural MLP não demonstram vantagem significante no uso de duas camadas ocultas ao invés de uma para problemas menores. Por isso, para a grande maioria dos problemas utiliza-se apenas uma camada oculta. Para a resolução de problemas de classificação, uma rede neural com uma camada oculta costuma ser mais que suficiente [37]. Portanto, optou-se por montar a rede com uma camada.

Número de Neurônios na Camada Oculta: Esse é um dos parâmetros que mais tende a variar para diferentes problemas, sendo em grande parte dos casos definido empiricamente. Deve-se ter cuidado em sua escolha, pois o uso de unidades demais pode gerar *overfitting*. Por outro lado, um número muito baixo de neurônios pode não ser capaz de generalizar de forma relevante os dados fornecidos. Inicialmente, optou-se pelo uso de 8 neurônios.

Taxa de aprendizado (Learning rate): Determina o modo como os pesos da rede são atualizados. Uma taxa de aprendizado muito baixa torna o aprendizado da rede muito lento, ao passo que uma taxa de aprendizado muito alta provoca oscilações no treinamento e impede a convergência do processo de aprendizado. Portanto, recomenda-se o uso de valores não muito altos. Foi escolhido o valor padrão da biblioteca utilizado: 0.001.

Função de ativação: Determina o nível de ativação do neurônio, o qual limita a amplitude do sinal de saída para uma faixa de valores finitos, usualmente normalizada entre [0,1] ou [-1,1]. Algumas das funções mais utilizadas são a sigmoide, tangente hiperbólica (tanh) e linear retificada (ReLU). A tanh, por permitir saídas positivas e negativas, torna-se uma alternativa atraente para servir de ativação às camadas ocultas. Para os neurônios da camada de saída foi usada a sigmoide, por poder ser interpretada como probabilidade.

Número de ciclos (epochs): Quantidade de vezes que os dados de treinamento passam pela rede antes do fim do algoritmo. Inicialmente definido com o valor 400.

Batch size: Hiperparâmetro de descida de gradiente que controla o número de amostras de treinamento a serem processadas antes que os parâmetros internos do modelo sejam atualizados. Foi usado inicialmente o valor 1, ou seja, para cada vez que uma amostra passasse pela rede, os pesos dos neurônios eram atualizados.

4.3.3 Treinamento da rede neural

Foi então dado início ao processo de treinamento. O *dataset* contém 1.048.575 registros referentes a fluxos de redes. Dado que esse é um número de amostras muito grande, optou-se pela utilização de validação por hold-out. Para esse tipo de validação, costuma-se selecionar a maior parte do *dataset* para treino, no entanto, optou-se pela utilização de apenas 20% do *dataset* para o conjunto de treino, de forma que 209.715 amostras iriam treinar a rede. Essa escolha deu-se devido ao tempo de treinamento e à hipótese de que esse seria um número suficientemente grande para que a ANN aprendesse a classificar os fluxos. Caso as predições mostrassem o contrário, o conjunto de testes seria reduzido.

A princípio, as *features* de treinamento foram aquelas descritas na seção 4.2. O processo de treinamento inicial e experimental foi realizado somente com essas *features*, sendo variados os valores dos hiperparâmetros. De acordo com o efeito das mudanças no resultado, optava-se por manter ou não cada configuração. Dessa forma, depois de diversos testes para diferentes parâmetros, a configuração definida foi a seguinte:

1. Função de ativação: mantida a ReLU para camadas ocultas e sigmoide para camada de saída.
2. Número de neurônios: ajustado para 16.
3. Número de camadas ocultas: mantido em somente uma.
3. Learning rate: mantido em 0.01.
5. Número de epochs: ajustado para 100.
6. Batch size: ajustado para 500.

4.3.4 Avaliação dos resultados obtidos

Ao se aumentar muito o número de neurônios ou o número de camadas, a precisão reduzia significativamente. Funções de ativação diferentes para a camada oculta produziam resultados semelhantes, no entanto, a ReLU se mostrou ser ligeiramente melhor. Nesse ponto, sem nenhuma otimização para as *features* selecionadas, a acurácia da ANN era de aproximadamente 68%.

A partir das configurações definidas, a próxima etapa foi encontrar as melhores *features*. Dentre todas as possíveis *features*, foram selecionadas aquelas que poderiam de alguma forma contribuir para a detecção de bots. Iniciou-se a análise de *features* promissoras como o tamanho do menor pacote enviado, tamanho máximo do bytes enviados em

um pacote, desvio padrão do tempo de envio de pacotes, entre outras. Diversos conjuntos de *features* foram montados para se avaliar quais produziriam melhores resultados de acurácia.

Foram encontradas *features* que quando usadas para o treinamento produziam 92% de acurácia para o conjunto de testes. Na tentativa de melhorar essa métrica, essas *features* foram fixadas para treino e outras *features* foram adicionadas 1 a 1 para que fosse percebida a melhora da ANN com a adição de cada nova informação. Em seguida, algumas outras variações do conjunto de *features* foram realizadas com base nos resultados encontrados na etapa anterior. Por fim, com o conjunto de *features* definido, cada *feature* foi removida individualmente para análise do impacto de sua retirada. Caso sua remoção não afetasse a capacidade de predição da rede, a *feature* era descartada.

Após um intenso conjunto de testes e remoção de *features* desnecessárias, foi obtido um conjunto que alcançava uma acurácia de 98%. As *features* selecionadas foram:

- **Fwd Pkt Len Max:** Tamanho máximo de pacote no sentido direto (bytes);
- **Bwd Pkt Len Max:** Tamanho máximo de pacote no sentido reverso (bytes);
- **Fwd IAT Max:** Máximo tempo entre 2 pacotes no sentido direto (ms);
- **Bwd IAT Max:** Máximo tempo entre 2 pacotes no sentido reverso (ms);
- **Init Fwd Win Byts:** Número de bytes enviados na janela inicial do sentido direto.

Para qualquer uma dessas *features* que fosse retirada, a acurácia da rede neural diminuía significativamente. Além disso, é um conjunto pequeno, o que é interessante para se reduzir custo computacional e evitar processamento desnecessário. Um número muito elevado de entradas poderia também levar a *overfitting*.

Para se ter uma ideia visual da diferença dos valores dessas *features* entre fluxos benignos e fluxos de bots, cada uma delas é apresentada na Figura 20, no formato de gráficos com 200 amostras aleatórias de cada classe. Dessa forma é possível ver as tendências e a variação dos valores que serviram de base para o aprendizado da ANN.

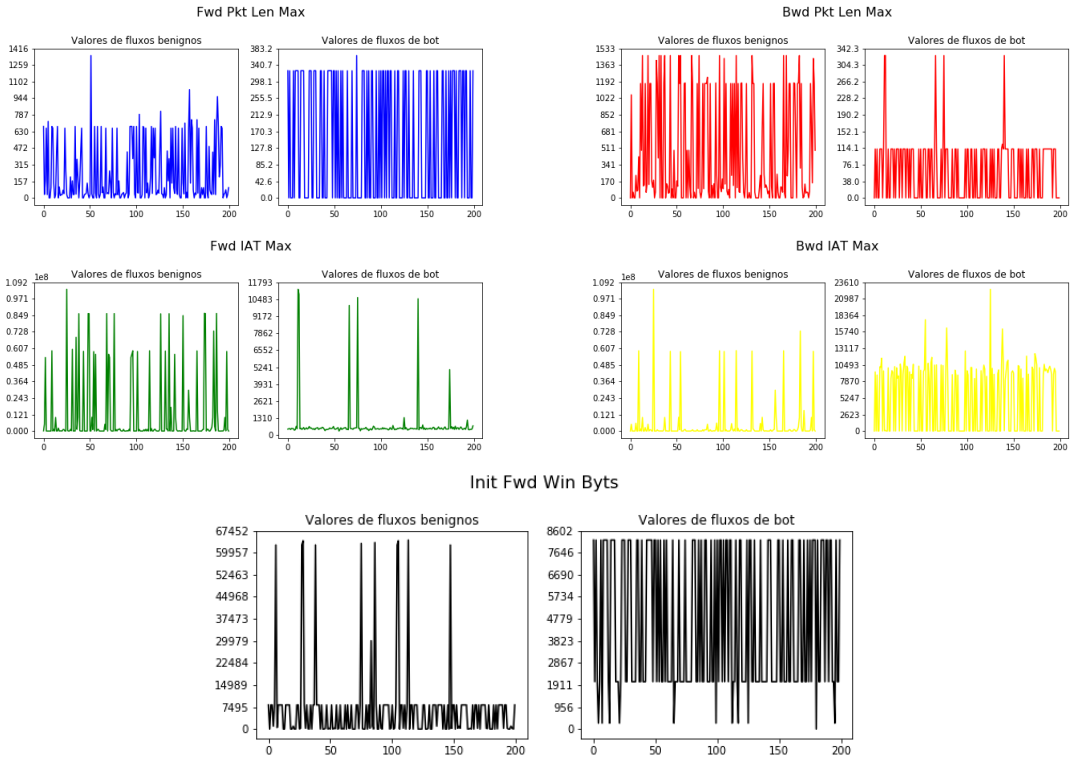


Figura 20 – Valores das *features* para fluxos benignos e de bots
Fonte: Elaborada pelos autores

Com relação ao tamanho de pacotes, para os fluxos benignos é possível ver que há uma variação muito maior do que para os fluxos de bots. Especialmente para a direção reversa, os valores benignos costumam ser muito mais altos, alcançando picos em 1460 pacotes, enquanto que para bots, dentre as amostras selecionadas, o maior valor é de aproximadamente 300 pacotes.

Já nos gráficos referentes ao tempo, é possível ver grandes variações nos fluxos das duas classes, porém, enquanto os fluxos de bot tendem a se manter em valores menores, os benignos alcançam valores altos com maior frequência. Além disso, os maiores valores são da ordem de 10^8, evidenciando que fluxos benignos costumam ficar muito mais tempo ociosos se comparados aos bots. Para o tamanho inicial da janela de transmissão, com exceção de alguns picos nos fluxos benignos, ambos os gráficos apresentam comportamento comparável.

Com relação ao treinamento da rede, os gráficos a seguir demonstram a evolução da acurácia e da perda com o passar das epochs. A função de custo utilizada é a função de entropia-cruzada, que calcula a diferença entre duas distribuições de probabilidades. A função de custo é usada para minimizar a perda, uma vez que quanto menor seu valor melhor é o modelo. É possível notar que cerca de 20 epochs já são suficientes para a acurácia e a perda alcançarem valores próximos do final. A partir daí, a variação passa a ser mais lenta.

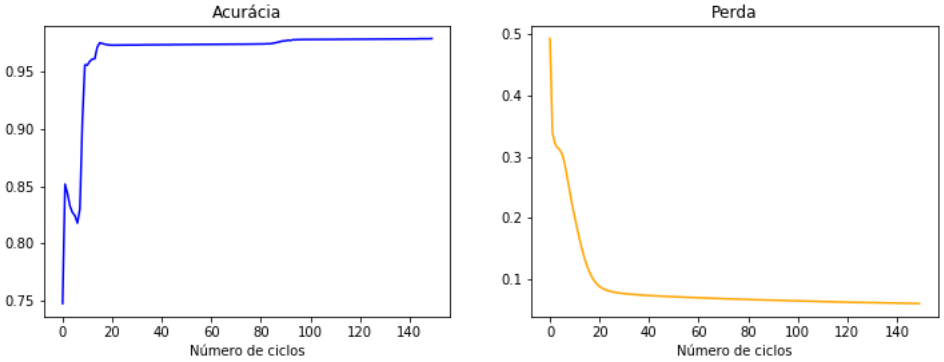


Figura 21 – Progresso do treinamento
Fonte: Elaborada pelos autores

Por fim, a matriz de confusão, juntamente com a as métricas descritas na seção 2.9.4, mostram o bom desempenho da rede para o *dataset* CIC-2018. O recall, que representa a capacidade da ANN de detectar bots é de 94% e o número de falsos negativos, que costuma ser um dos principais problemas em sistemas de detecção, foi de apenas 549 fluxos (0.09% dos fluxos benignos).

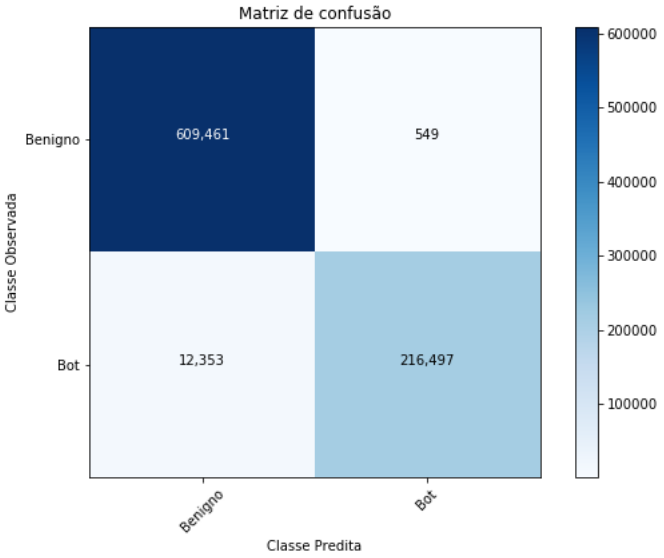


Figura 22 – Matriz de confusão para o *dataset* CIC-2018
Fonte: Elaborada pelos autores

Métrica	Valor (%)
Acurácia	98,46
Precisão	99,75
Recall	94,60
F1-Score	96,49

Tabela 2 – Métricas para o *dataset* CIC-2018

Durante o processo de definição dos hiperparâmetros da ANN, foi notado que o aumento do *batch size* reduzia drasticamente o tempo de treinamento de rede. Para efeito comparativo, a tabela a seguir demonstra a acurácia, o recall e o tempo de treinamento da rede para 100 epochs ao se variar o *batch size*.

Tamanho do Batch	Acurácia (%)	Recall (%)	Tempo (segundos)
500	97.79	94.28	134.78
400	97.82	94.31	191.26
300	97.85	94.42	266.17
200	97.97	94.57	333.28
100	97.99	94.58	395.97
50	98.01	95.53	754.57

Tabela 3 – Acurácia ao se variar o batch size

Percebe-se que ao se utilizar um *batch size* de 500 ou um *batch size* dez vezes menor, a acurácia e o recall são bastante similares, no entanto, o tempo que se leva com a redução desse parâmetro é 6 vezes menor. Dado que a redução de tempo praticamente não compromete a precisão encontrada ao final, durante o processo de otimização optou-se por valores de *batch size* maiores para acelerar o processo.

Com os resultados obtidos, a próxima etapa seria testar a ANN treinada para diferentes *datasets*.

4.4 Avaliação da ANN montado para predição em outros *datasets*

Após realizada a predição, buscaram-se outros *datasets* de teste para uma melhor avaliação do modelo obtido. Pretendia-se, primeiramente, testar a rede neural em um conjunto de dados obtidos a partir de um ambiente de simulação parecido com o testado no *dataset* CIC-2018. Por essas razões, foi escolhido o *dataset* CIC-2017. Esperava-se que, por conta de ambos os *datasets* terem sido construídos por pesquisadores da mesma universidade sob condições semelhantes, a rede obtivesse boas métricas de predição também para o *dataset* de 2017.

No entanto, não foi o que ocorreu: os resultados obtidos para o novo *dataset* foram muito ruins. Apesar de possuir uma alta acurácia - 0.9591 - ao analisar a matriz de confusão do teste, percebeu-se que a rede conseguia identificar a maioria do tráfego benigno, mas não conseguia detectar a maioria do tráfego de bot, tendo, portanto, uma quantidade de falsos negativos muito alta. Ou seja, uma grande quantidade de fluxos de bot foi classificada como benigno.

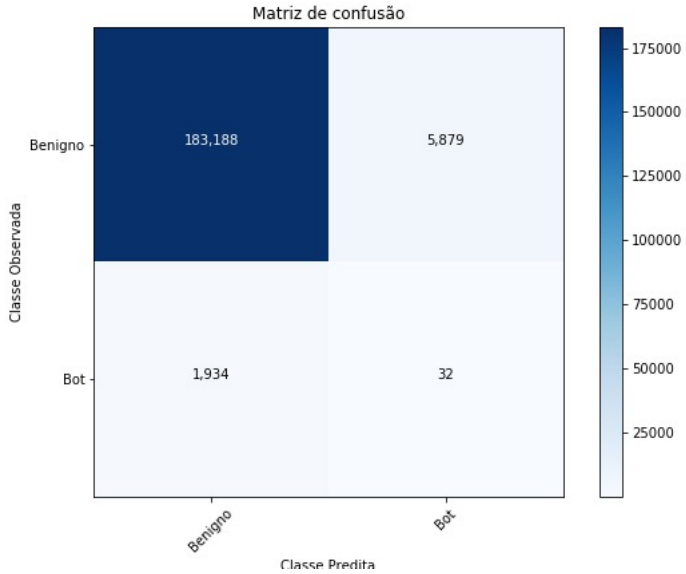


Figura 23 – Matriz de confusão para o *dataset* CIC-2017
Fonte: Elaborada pelos autores

O baixo desempenho da rede, muito provavelmente advém do fato de os bots usados em cada uma das simulações serem diferentes. Enquanto o *dataset* de 2018 utiliza como predominante a botnet Zeus, o *dataset* de 2017 utiliza somente a botnet Ares.

A qualidade da ANN também foi avaliada para os *datasets* da CTU. Para extrair as *features* necessárias dos 13 *datasets*, fez-se necessário o uso do CicFlowMeter nos arquivos pcap. Por questões de privacidade, os fluxos referentes a pacotes benignos não se encontram disponíveis publicamente. Por isso, somente seria possível analisar a ANN para os fluxos de bot, o que é suficiente para avaliar a capacidade da rede de identificar fluxos maliciosos. Devido à existência de uma única classe, a métrica usada para avaliação foi somente a acurácia, que nesse caso é igual ao recall.

Novamente, obteve-se um resultado de predição muito ruim: a rede neural continuava sem conseguir identificar os fluxos de bot. Devido a isso, algumas outras tentativas de treino com hiperparâmetros e *features* diferentes foram feitas, mas nenhuma com sucesso.

Dataset CTU	Corretamente classificados (TP)	Incorretamente classificados (FN)	Acurácia (%)
42	75	10443	0,71
43	57	8968	0,63
44	2	10189	0,02
45	0	159	0
46	34	1048	3,14
47	102	386	20,9
48	1	51	1,92
49	0	4777	0
50	1898	67294	2,74
51	1	5468	0,02
52	0	230	0
53	26	6241	0,41
54	225	31980	0,7

Tabela 4 – Acurácia para o *dataset* CTU-13

Analisando visualmente os valores das *features* dos *datasets* estudados, percebeu-se que eles eram bastantes diferentes entre si e este poderia ser o motivo pelo qual a rede neural não conseguia identificar os fluxos de botnet em *datasets* diferentes do de treinamento. Para os fluxos de bots, a *feature* "Fwd Iat MAX" no *dataset* CIC-2018 raramente ultrapassava o valor 750, enquanto que para o *dataset* CIC-2017 os valores estavam comumente na casa de 10^7. O mesmo ocorria para diversas *features* dos *datasets* da CTU: as diferenças entre valores muitas vezes eram absurdamente grandes. De fato, isso é algo que se pode esperar de diferentes botnets, contudo, a acurácia da rede nesses casos foi tão ruim que não se podia dizer que ela generalizava aspectos intrínsecos de uma botnet.

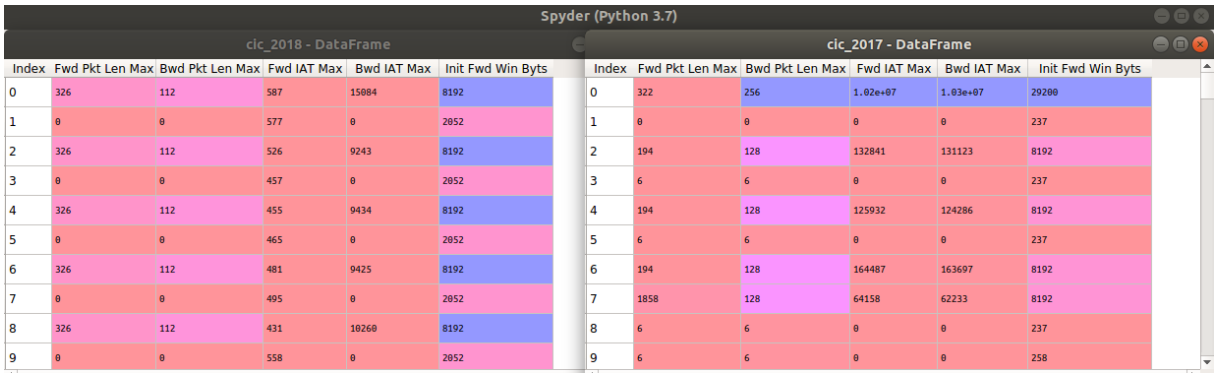


Figura 24 – Amostras das features
À esquerda: Amostras das *features* do *dataset* CIC 2018
À direita: Amostras das *features* do *dataset* CIC 2017
Fonte: Elaborada pelos autores

Para tentar verificar as semelhanças entre os *datasets* de uma forma mais sistemática, escreveu-se um algoritmo que calculava o valor da correlação entre as *features* do *dataset* CIC-2018 em relação ao *dataset* CIC-2017 e os vários *datasets* da CTU. Os valores para as correlações foram gerados por meio do método 'corrwith' da biblioteca pandas [38], sendo usado coeficiente de correlação de Spearman.

É nítido que a correlação para praticamente todos os *datasets* é próxima de zero, ou seja, as variáveis são praticamente descorrelatadas. Os maiores valores de correlação são para o *dataset* CIC-2017, na faixa de 0,4, o que também não indica uma forte correlação.

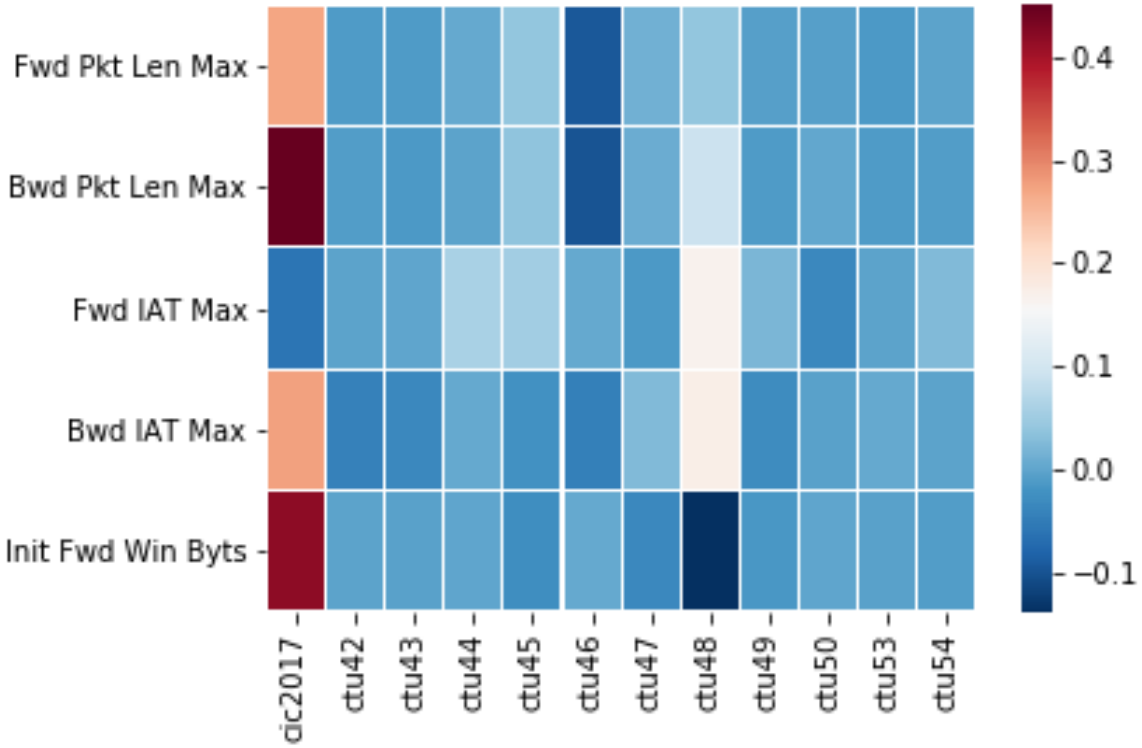


Figura 25 – Correlação entre as *features* selecionadas para os diferentes *datasets*
Fonte: Elaborada pelos autores

4.5 Simulação própria

Dado que os *datasets* poderiam ser tão distintos pelo simples fato de utilizarem botnets diferentes, decidiu-se realizar a simulação de uma botnet Ares, a mesma usada pelo *dataset* CIC-2017, para comparar os valores das *features*.

Utilizou-se o código fonte de uma botnet Ares - escrito em Python e disponibilizado em [39] - para realizar as devidas simulações. Além disso, para construir o novo *dataset*, usou-se uma máquina virtual operando com a distribuição Kali Linux (release 2019.4) como atacante e uma máquina virtual com Windows 10 como a vítima do ataque. Ademais,

as duas máquinas virtuais foram configuradas em uma mesma sub-rede para que fosse possível a comunicação entre ambas.

Com o ambiente de simulação pronto, foi necessário primeiramente subir um serviço HTTP em Python para servir como C&C da botnet operando no Kali. A figura abaixo mostra o painel de controle da máquina atacante.

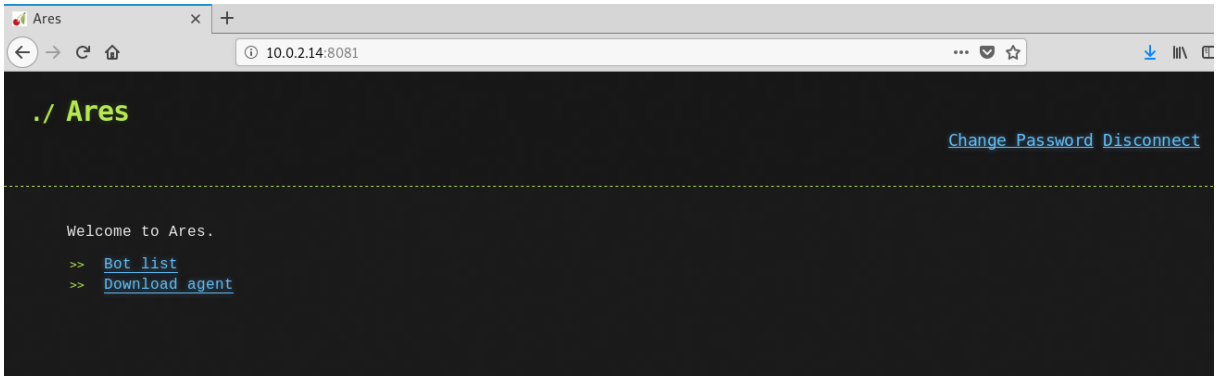


Figura 26 – Painel Central da Unidade de Comando e Controle
Fonte: Captura de tela do C&C

A partir da imagem, é possível perceber que existe a opção de verificar a lista de bots e realizar o download do agente - arquivo que será executado no computador da vítima. Com o C&C pronto, agora é necessário infectar a vítima para que seja possível realizar a construção da botnet. Isso pode ser feito de diversas maneiras, como por exemplo, utilizando-se do ataque man-in-the-middle ou fazer com que a vítima realize o download do agente da botnet junto a um arquivo desejado. Entretanto, como o objetivo da simulação é apenas capturar o tráfego da comunicação entre o bot e o C&C, apenas realizou-se o download do arquivo do agente da botnet Ares diretamente da máquina virtual da vítima. Foi necessário executar o agente da Ares para o computador da vítima se conectar ao C&C e, dessa forma, comportar-se como um bot. Dessa forma, o computador da vítima realiza a conexão com o servidor HTTP da unidade central e fica suscetível a ataques.

Antes de realizar o ataque, iniciou-se a captura de pacotes por meio do Wireshark no computador da vítima para registrar todos os pacotes da comunicação. A Figura 27 mostra a alteração no painel de controle do C&C, informando que existe um hospedeiro infectado.

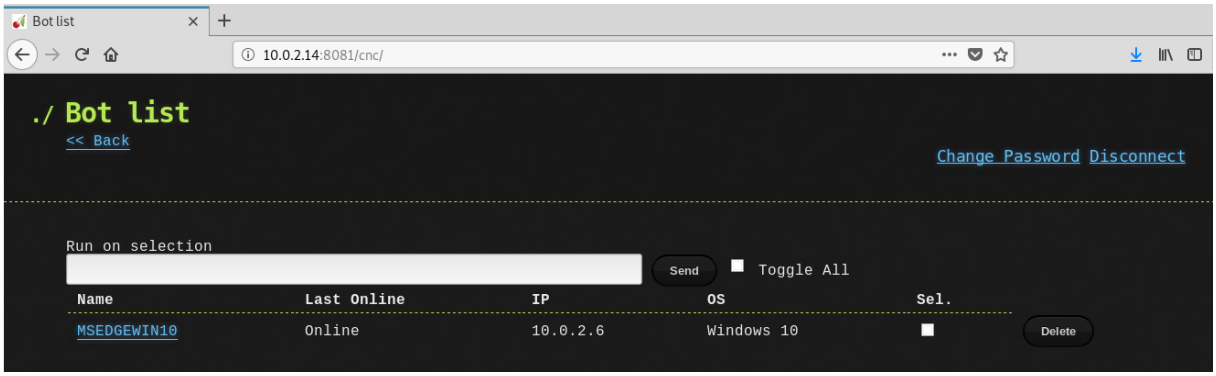


Figura 27 – Listagem de bots suscetíveis a ataques
Fonte: Captura de tela do C&C

Finalmente, com todo o ambiente preparado para a realização dos ataques, iniciou-se a simulação. Por meio do C&C, foram realizadas diversas operações no computador da vítima, como capturas de tela, criação e exclusão de arquivos por meio do terminal e uploads de arquivos armazenados no sistema operacional. Essas operações foram realizadas diversas vezes e intercaladas com acesso a websites, pois também havia o intuito de capturar fluxos benignos. A figura abaixo mostra a obtenção da captura de tela do computador da vítima e a abertura de aplicativos por meio de comandos realizados do Kali Linux.

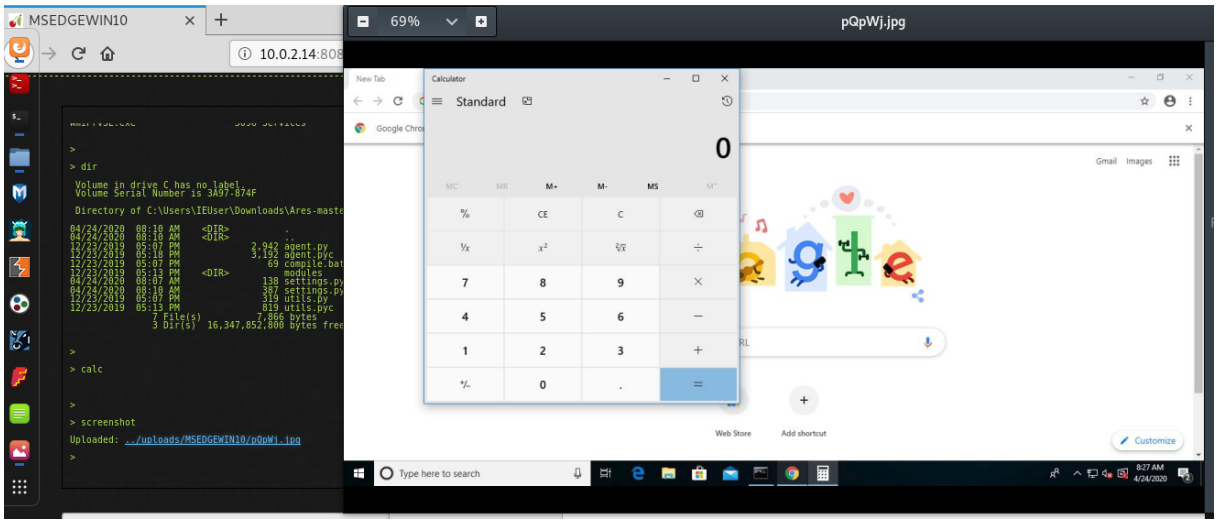


Figura 28 – Ataques realizados sobre o computador da vítima
Fonte: Captura de tela do C&C

Com a captura realizada, foi necessário apenas utilizar o CicFlowMeter para retirar as *features* do arquivo pcap gerado para efetivar a criação do *dataset*.

Após a finalização de todo o processo de geração do novo *dataset*, testou-se a rede neural com o novo conjunto de dados obtido, mas novamente o resultado foi ruim. Ao se analisar o valor das *features*, foi possível perceber que estas também se diferenciavam bastante dos demais *datasets*.

Com isso foi possível notar que, mesmo analisando somente a botnet Ares, os valores das *features* eram demasiado distintos para simulações diferentes. Fica claro que sistemas de detecção por *features* são muito sensíveis às condições em que a botnet opera e as configurações disponíveis para uma mesma botnet. Um atacante pode variar diversos parâmetros de conexão e fluxos na tentativa de burlar sistemas de detecção que sejam baseados nesses parâmetros. Dadas as conclusões acima apresentadas, a análise por *features* foi deixada de lado e buscou-se encontrar um método que fosse menos sensível a tais variações.

5 Análise Baseada em Grafos

Como a análise por meio das informações de fluxo de comunicação não apresentou resultados satisfatórios, buscou-se utilizar algum tipo de método diferente. Após algum estudo, optou-se por uma abordagem baseada em grafos. Apesar de muitos métodos de detecção de botnets serem feitos por meio da análise do fluxo de dados, esse método só consegue identificar as características de um tráfego de bot com base em links individuais, e não na estrutura da rede como um todo. Normalmente, esse tipo de análise requer que a botnet produza valores semelhantes àqueles com os quais o modelo de detecção foi treinado para poder detectar os tráfegos maliciosos. Essa característica faz com que o atacante tenha a liberdade de realizar pequenas e até mesmo simples modificações no comportamento da botnet para passar despercebido pelo IDS. Pode-se tomar como exemplo a utilização de criptografia de comprimento variável ou até mesmo a variação do tamanho dos pacotes enviados. Somente essa última variação dificultaria bastante a detecção pela rede neural construída anteriormente. A análise baseada em grafos dispensa a necessidade de realizar comparações de *features* entre os fluxos de comunicação e passa a analisar a rede de comunicação como um todo.

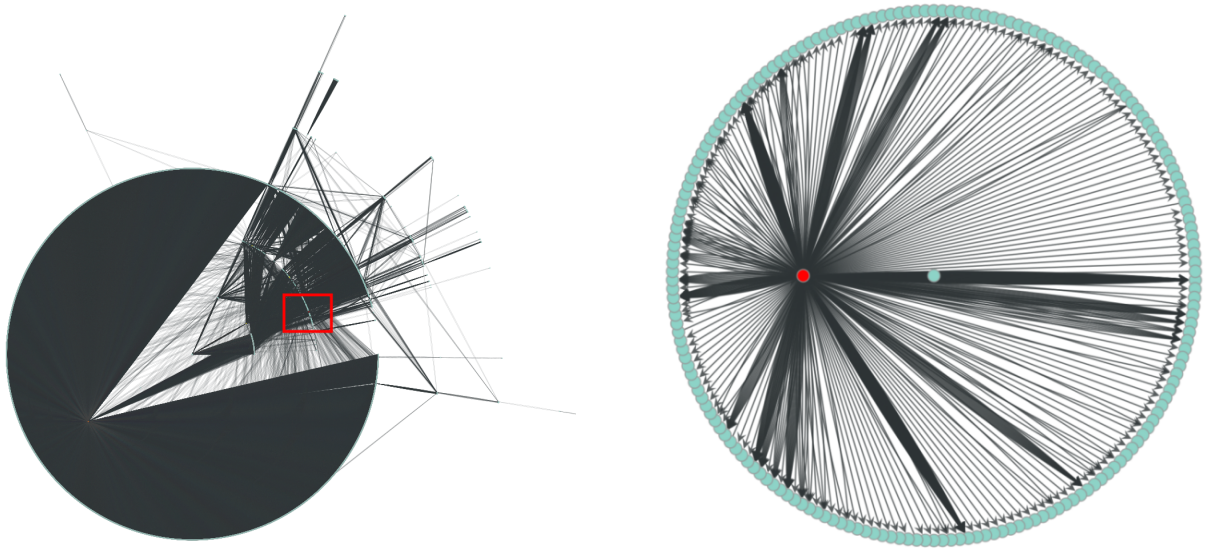
Existem diversos estudos que utilizam abordagens por grafos para detecção de botnets, como em [40] e [41]. Por meio de métricas e características do grafo que representa a simulação, são exploradas as relações espaciais do tráfego gerado.

5.1 Construção do grafo e remodelagem dos *datasets*

O grafo representativo dos fluxos de rede foi montado de forma que cada IP fosse representado como um nó, enquanto que cada conexão entre dois nós caracterizava uma aresta direcionada. Para múltiplas conexões entre as mesmas duas máquinas, apenas uma aresta foi criada. Após extrair as *features* dos *datasets* com o CicFlowMeter, esses dados eram passados ao 'graph-tool', que é um módulo otimizado capaz de construir grafos e calcular diversas métricas sobre eles. Essa biblioteca é utilizada via Python, no entanto, suas funções e estruturas núcleo são implementadas em C++, fazendo extenso uso do template metaprogramming, que é baseado na bilbioteca Boost Graph. Isso confere um nível de performance que é comparável (tanto em memória, quanto em processamento) a uma biblioteca C/C++ pura [42].

Os grafos foram gerados para os dois *datasets* do CIC e todos os treze *datasets* da CTU. As imagens a seguir mostram, respectivamente, o grafo completo gerado para o *dataset* 46 da CTU e um trecho do grafo que representa somente as conexões do único bot desse *dataset*. Em ambos os grafos, plotados em esquema circular, os círculos azuis

representam os nós benignos e o círculo vermelho indica o nó do bot. Na Figura 29a, a localização do bot está destacada pelo quadrado vermelho.



(a) Grafo para o dataset inteiro (b) Grafo somente para as conexões do bot

Figura 29 – Grafo gerado para o *dataset* CTU-46
Fonte: Elaborada pelos autores

5.2 Preparação dos dados

5.2.1 Seleção de *features*

Com base no trabalho de Chowdhury et al. [41], optou-se por selecionar *features* que representassem bem as relações dos nós com seus vizinhos e o grau de centralidade de cada nó. Visto que é comum botnets estabelecerem conexões com diversos outros pares, esse tipo de característica poderia ser conveniente para a detecção. As *features* utilizadas foram as seguintes:

Grau de entrada (*in degree*): Definido como a quantidade de fluxos que são estabelecidos de um nó vizinho para o nó em questão. Quanto maior o número de máquinas que se comunicam com um nó, maior será o valor dessa *feature*. Uma vez que um servidor C&C tende a receber fluxos de todos os nós que comanda, um alto grau de entrada deve ocorrer.

Grau de saída (*out degree*): Definido como a quantidade de fluxos que são estabelecidos de um dado nó para seus vizinhos. Um alto valor indica que o nó tende a se comunicar com diversas outras máquinas. Como bots tendem a abrir conexões com outras potenciais vítimas na tentativa de infectá-las, o grau de saída pode ser um bom indicativo para apontar a presença de bots no grafo.

Peso de entrada (*in degree weight*): Refere-se à quantidade de dados que chegam ao nó por meio de seus vizinhos, podendo ser representado, por exemplo, pelo número de bytes captados. Está diretamente relacionado com a quantidade de pacotes recebidos, os protocolos utilizados, os cabeçalhos dos pacotes e o tamanho das mensagens. Botnets de uma mesma classe usualmente apresentam comportamento similar na transferência de informações e envio de comandos. Portanto, é plausível supor que bots devem receber, aproximadamente, o mesmo volume de dados nas comunicações entre si e com o servidor C&C.

Peso de saída (*out degree weight*): Refere-se à quantidade de dados que um nó envia aos seus vizinhos. Segue a mesma linha de raciocínio do peso de entrada, no entanto, para o sentido reverso.

Centralidade de intermediação (*betweeness centrality*): Na teoria dos grafos, a centralidade de intermediação entre nós quantifica o número de vezes que um nó atua como uma ponte ao longo do caminho mais curto entre dois outros nós. Vértices com alta intermediação podem ter uma influência considerável dentro de uma rede em virtude de seu controle sobre a passagem de informações ao longo do grafo. Eles também são aqueles cuja remoção da rede terá mais impacto nas comunicações entre os demais vértices, porque estão no maior número de caminhos percorridos pelas mensagens [43]. A centralidade C_v de um nó v pode ser matematicamente descrita como:

$$C_v = \sum_{v \neq u \neq w} \frac{\sigma_{u,w}(v)}{\sigma_{u,w}} \tag{5.1}$$

Em que $\sigma_{u,w}$ é a quantidade de menores percursos de u a w e $\sigma_{u,w}(v)$ é a quantidade de menores percursos de u a w que passam por v.

Coeficiente de agrupamento local (*local clustering coefficient*):O coeficiente de agrupamento é uma medida do grau em que os nós tendem a se agrupar. As evidências sugerem que, na maioria das redes do mundo real, os nós tendem a criar grupos fortemente unidos, caracterizados por uma densidade relativamente alta de vínculos; essa probabilidade tende a ser maior que a probabilidade média de uma ligação estabelecida aleatoriamente entre dois nós (Holland e Leinhardt, 1971; Watts e Strogatz, 1998). Existem duas versões dessa medida: a global e a local. O coeficiente local C_v para um vértice v é uma medida do grau de conexão entre os nós dentro de sua vizinhança dividida pelo número de links que poderiam existir entre eles. Pode ser representado como:

$$C_v = \frac{e_v}{K_v(K_v - 1)} \tag{5.2}$$

sendo e_v é o número de pares conectados entre os (K_v) vizinhos de v.

Centralidade do vetor próprio (*eigen vector centrality*): É uma métrica que mede o grau de influência que um nó tem na rede. Nem todos os vértices de um grafo são equivalentes: alguns são mais relevantes que outros baseado no peso das arestas, portanto, esses nós devem ter peso maior [44]. Não necessariamente um nó com muitos links tem uma centralidade de vetor próprio alta. Além disso, um nó com um valor alto para essa *feature* não necessariamente possui muitos links. Um nó que possua algumas poucas conexões com outros nós relevantes tem um alto valor de centralidade, seguindo essa métrica.

Seja:

$$a_{v,w} = \begin{cases} 1 & \text{se o nó v é linkado ao nó w} \\ 0 & \text{se o nó v não é linkado ao nó w} \end{cases} \tag{5.3}$$

A Centralidade do vetor próprio do nó v (x_v) pode ser definida como:

$$x_v = \frac{1}{\lambda} \sum_{wpertenceM(v)} a_{v,w} x_w \tag{5.4}$$

em que M(v) é o conjunto de nós do nó v e λ é uma constante.

5.2.2 Extração das *features* selecionadas

Após a geração dos grafos, as *features* de centralidade e direcionalidade acima apresentadas foram extraídas por meio da própria biblioteca graph-tool. Vários *datasets* possuíam tamanho muito grande, chegando o maior deles a mais de 600 MB. Para exemplificar, o *dataset* 48, que é o segundo menor, possui 15,6 MB, porém, possui 38.203 IPs distintos, o que implica no mesmo número de nós. Por isso, o custo computacional para o cálculo das *features* dos grafos é muito alto. As *features* dos menores grafos eram calculadas em questão de minutos, enquanto que, para os maiores, levavam-se horas. Todos foram gerados em um notebook comum de core i5 e 8GB de memória RAM e o tempo levado para o cálculo de todas as *features* encontra-se na tabela a seguir.

Dataset	Número de Nós	Número de Arestas	Tempo Total (minutos)
CTU-42	607565	5649272	4087
CTU-43	442471	3616244	1655
CTU-44	434988	9421276	3365
CTU-45	186244	2242152	489
CTU-46	41658	259664	9
CTU-47	107341	1117838	66
CTU-48	38204	228154	5
CTU-49	383788	5908460	2430
CTU-50	367263	4175016	1413
CTU-51	197824	2619582	324
CTU-52	41931	214502	8
CTU-53	94434	650942	59
CTU-54	315769	3850298	1283

Tabela 5 – Tempo necessário para criação dos grafos

De posse das *features* já calculadas, uma nova estrutura de dados - no caso, um dataframe - foi gerado para análise posterior.

5.2.3 Filtragem de nós

Assim como em [41], as *features* de grafos acima descritas foram utilizadas para alimentar um mapa auto-organizável (SOM).

O SOM utilizado foi uma implentação minimalista, e baseada na biblioteca numpy do Python, chamada MiniSom, que pode ser encontrada em [45]. A saída do SOM é um mapa em duas dimensões, no qual amostras semelhantes são posicionadas próximas umas das outras. O mapa é construído por meio de um técnica conhecida como Best Matching Unit (BMU), que calcula a distância de cada peso (referente a um nó) em relação a um vetor de entrada. O peso com menor distância é denominado vencedor. Existem diversas maneiras de se calcular essa distância, sendo a mais comum - e que foi utilizada - por meio da distância euclidiana. Caso a distância média seja alta, os pesos dos nós ao redor são muito diferentes, fazendo com que esta região possua uma cor mais clara no mapa. Entretanto, caso a média seja baixa, o contrário acontece. A figura a seguir apresenta um exemplo visual dessa técnica, a qual permite visualizar onde a concentração de diferentes clusters são mais predominantes.

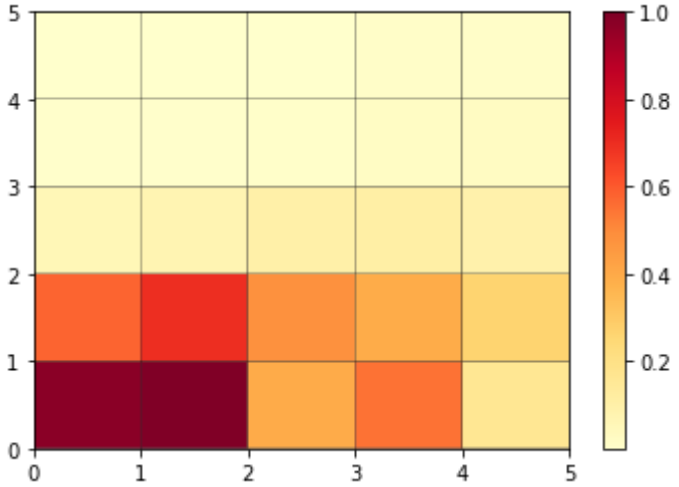


Figura 30 – Mapa de Densidade para o *dataset* CTU-54
Fonte: Elaborada pelos autores

A ideia por trás do SOM era de que fluxos benignos apresentam tráfegos muito semelhantes, portanto, o SOM os agruparia nos mesmos clusters. Seria esperado que se formasse um ou alguns clusters com a grande maioria dos nós (benignos), enquanto que os demais clusters se formassem com poucos nós de comportamento anormal, estando dentro desses clusters os nós correspondentes aos bots. Isso permitiria eliminar do *dataset* a maior parte das máquinas não suspeitas, reduzindo significativamente o número de nós a serem analisados e o custo computacional.

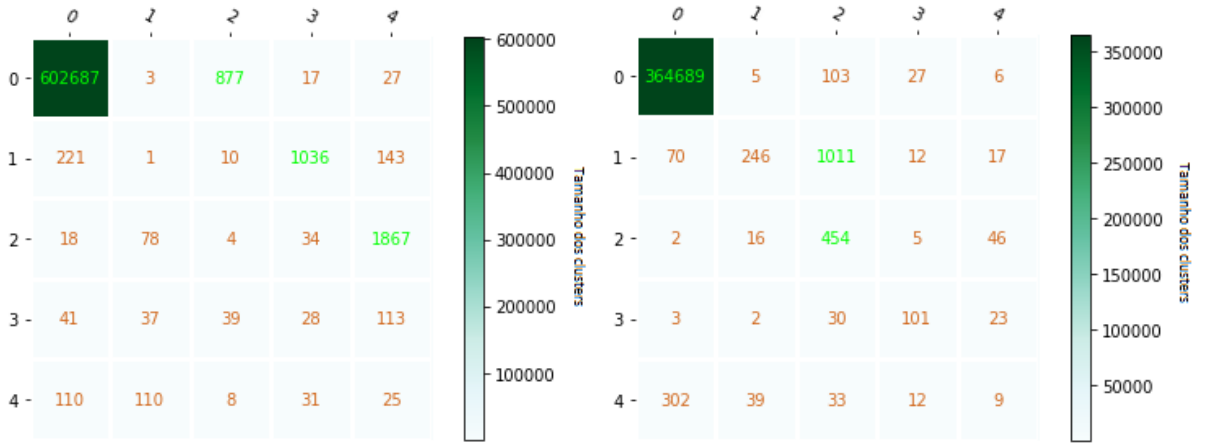
Todos os nós agrupados nos maiores clusters seriam considerados benignos. No entanto, deve-se notar que bots também poderiam ser incorretamente agrupados nos maiores clusters, o que faria com que fossem incorretamente filtrados e excluídos antes mesmo da etapa de classificação. Por isso, além de maximizar o tamanho dos clusters benignos para maior filtragem de nós, é importante minimizar o número de bots presentes nesses clusters. Para que isso fosse possível, foi necessário ajustar os hiperparâmetros mostrados a seguir:

- **A dimensão (AxB) do SOM:** Dimensão que será usada para construção do Mapa de Kohonen.
- **A taxa de aprendizado:** Valor entre 0 e 1 o qual controla a rapidez com que os pesos são atualizados.
- **Sigma:** A disseminação da função de vizinhança, definida como: $sigma(t) = \frac{sigma}{(1+t/T)}$, em que T é metade do número da iteração.
- **Função de decaimento (opcional):** usada para reduzir a taxa de aprendizado e o valor de sigma a cada iteração.

Após o ajuste dos parâmetros, foi escolhido construir um mapa de dimensão 5x5 com taxa de aprendizado de 0.7 e sigma 0.5, sem função de decaimento. Com esse ajuste foi possível perceber que o algoritmo conseguia agrupar a grande maioria dos nós benignos em um único cluster, fazendo com que os nós restantes (tanto benignos quanto bots) fossem agrupados nos clusters remanescentes, mas cada um destes com um tamanho muito menor. Com isso, era necessário apenas escolher o tamanho mínimo dos clusters que deveriam ser descartados para realizar a filtragem dos nós. Esse tamanho mínimo (limiar) foi escolhido por meio da fórmula:

$$T = \frac{\sqrt{S}}{2} \tag{5.5}$$

Em que T representa o limiar escolhido e S representa o número total de nós do *dataset* analizado. A Figura 31 mostra a distribuição do número de nós em cada um dos 25 clusters (cada quadrado representa um cluster). Os números em verde indicam os clusters que atingiram o valor mínimo de limiar e, portanto, devem possuir apenas nós benignos.



(a) *Dataset* CTU-42 (b) *Dataset* CTU-50

Figura 31 – Quantidade de nós por cluster
Fonte: Elaborada pelos autores

É possível observar que com o limiar escolhido foi possível eliminar uma grande parte do número de nós do *dataset*, permanecendo apenas os possíveis clusters de bot, os quais tinham um tamanho menor. A Figura 32 mostra a porcentagem de nós filtrados para cada *dataset* da CTU.

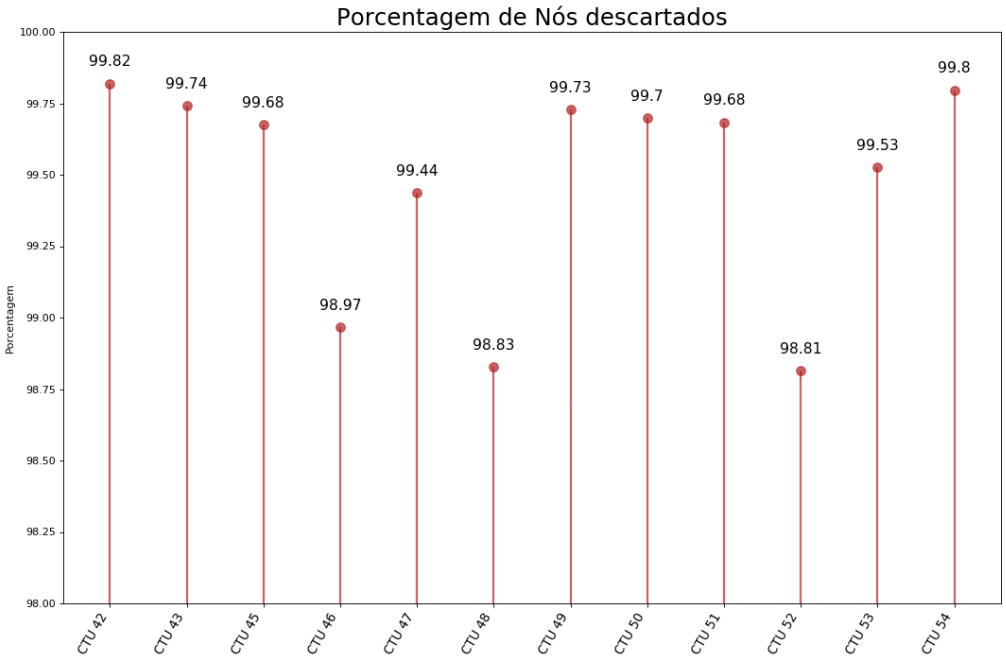


Figura 32 – Porcentagem de nós filtrados
Fonte: Elaborada pelos autores

5.3 Classificação

5.3.1 Método usado

Com os nós filtrados, é necessário finalmente utilizar um método para classificar quais nós correspondiam a bots. Entretanto, para realizar tal classificação, é necessário identificar primeiramente o endereço IP de cada nó para verificar se corresponde a um bot ou não. Assim, foi necessário inserir no conjunto de dados utilizados um identificador para cada nó. Uma vez que, por padrão, na biblioteca todas as entradas são normalizadas e utilizadas para treino, foi necessário fazer um pequeno ajuste na biblioteca para que a coluna do dataframe referente ao endereço IP fosse mantida inalterada e não entrasse no treinamento. Além disso, foi possível observar que nenhum nó de bot foi agrupado em um cluster com tamanho maior que o limiar, ou seja, nenhum bot foi descartado na etapa de filtragem.

Com isso, todo o processo para classificação dos nós se dá por uma metodologia híbrida, onde é utilizado o SOM para filtragem de nós (algoritmo não supervisionado) e um método supervisionado de aprendizado de máquina para a categorização dos nós. Pode-se dizer que a identificação dos bots é realizada em três etapas, como mostrado no diagrama a seguir.

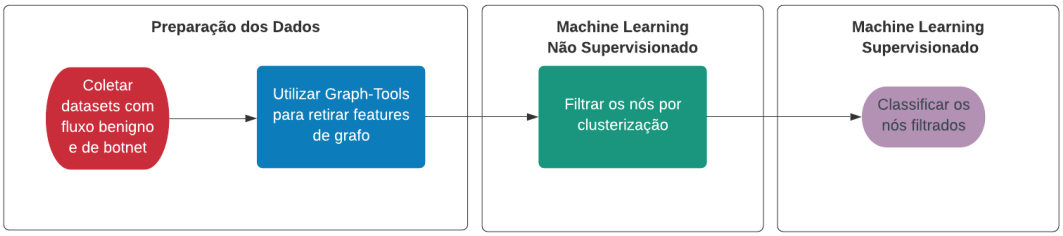


Figura 33 – Diagrama das etapas realizadas
Fonte: Elaborada pelos autores

Para a classificação dos nós, utilizou-se o algoritmo Decision Tree por apresentar vantagens como menor consumo computacional, não necessidade de normalização e maior facilidade de construção. Com isso, extraíram-se as *features* de todos os *datasets* CTU-13 e CIC-2017 e juntaram-se todas as informações retiradas em um único dataframe. Não foi possivel utilizar o CIC-2018 pelo fato deste não possuir as informações referentes aos endereços IP de origem e destino de cada fluxo.

As próprias informações do grafo foram utilizadas para realizar o treinamento da decision tree. Devido ao baixo número de nós após a filtragem e o baixo número de nós correspondentes a bots, a abordagem para avaliação do método de detecção foi através de validação cruzada. O *dataset* foi divido igualmente em 10 partições, em que cada uma era usada como teste durante cada iteração. Cada partição continha, necessariamente, 3 ou 4 nós de bot.

5.3.2 Análise de resultados

Após realizada a preparação dos dados e o treinamento do sistema de classificação, as métricas para a avaliação da qualidade do modelo são apresentadas na Tabela 6.

Partição	Acurácia (%)	Precisão (%)	Recall (%)	FPR (%)
1	99,8	75	75	0,10
2	99,8	75	75	0,10
3	99,59	50	75	0,30
4	99,8	100	50	0
5	99,9	80	100	0,10
6	99,8	66,67	66,67	0
7	99,8	100	33,33	0
8	99,9	75	100	0,10
9	99,59	42,86	100	0,41
10	99,49	33,33	33,33	0,20
Média	99,75	69,79	70,83	0,13

Tabela 6 – Métricas para os *datasets* da CTU

Pela tabela, é possível ver que o método utilizado possui alta acurácia para todas as partições, já o recall sofre maior variação. Para as partições com recall igual a 0.75, significa que 3 dos 4 bots presentes foram identificados e para as partições com recall igual a 66,67, 2 dos 3 bots foram identificados. Os piores casos foram das partições 7 e 10, em que dos 3 bots apenas 1 foi identificado. Chamam a atenção os valores referentes ao número de falsos positivos: o modelo praticamente não classifica nós benignos como sendo bots. Essa característica é extremamente importante, visto que um dos principais problemas em sistemas de classificações é a grande quantidade de alertas sem significado.

Vendo o resultado médio, nota-se que o modelo de detecção consegue detectar cerca de 70% dos bots com um índice de falsos positivos muito baixo. Sabendo que as botnets do CTU-13 e do CIC são de tipos diferentes e com características bastante distintas, o modelo apresenta um índice de detecção robusto com baixa quantidade de alertas incorretos. Portanto, a detecção se mostra não atrelada a um tipo específico de botnet nem a condições específicas de tráfego.

A Figura 34 mostra o fluxograma da árvore de decisão, onde é possível perceber os pesos de cada *feature* e os coeficientes calculados para cada uma. Além disso, analisando a figura, pode-se perceber que as *features* mais importantes para a classificação de cada nó foram *out degree*, *node betweeness* e *in weight*.

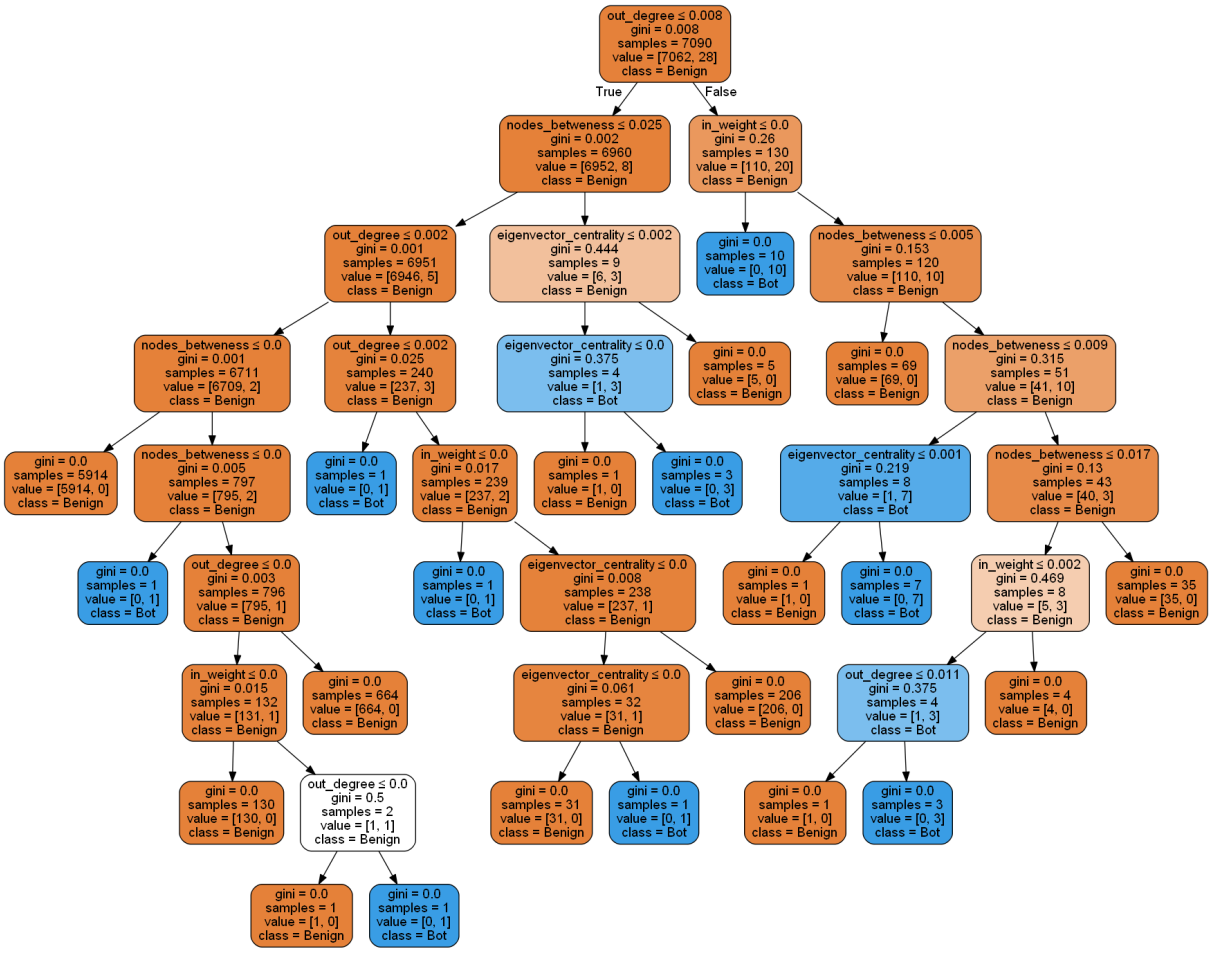


Figura 34 – Fluxograma da árvore de decisão
Fonte: Elaborada pelos autores

Conclusão

Foram apresentados dois tipos de abordagem para a análise e detecção de botnets presentes na rede. A primeira, baseada em fluxos, apresentou ótimos resultados para um *dataset* em específico, no entanto, quando exposta a outras amostras de teste não demonstrou performance efetiva. Portanto, esse método não foi capaz de realizar a detecção de diferentes tipos e variações de bots frente aos fluxos benignos. Isso decorre do fato de o sistema ter se tornado muito específico, sendo apropriado o suficiente para capturar os padrões presentes na botnet de treino, mas não competente o bastante para generalizar outras características de botnets.

Já a segunda abordagem, baseada na análise do tráfego de redes como grafos direcionados, é dividida em duas etapas. A primeira é responsável pela filtragem, eliminando a grande maioria dos nós com tráfego similar e majoritariamente benigno, enquanto que a segunda aplica a classificação propriamente dita sobre cada unidade da rede. Pelos resultados, foi possível perceber que a primeira etapa foi altamente eficiente em separar os nós não suspeitos, diminuindo a o processamento necessário na etapa de identificação dos dispositivos pertencentes à botnet.

Por fim, a classificação feita na segunda etapa demonstrou alto recall com baixo número de falso positivos, indicando que esse IDS pode detectar boa parte dos bots a ele apresentados com baixa quantidade de falsos alertas. Os resultados foram avaliados em diferentes *datasets*, com diferentes tipos de botnets em cada um, o que indica que o sistema é capaz de identificar padrões intrínsecos do malware tratado, não estando atrelado às características e configurações específicas de uma única variação.

Referências

[1] BRASIL sofreu 15 bilhões de tentativas de ataques cibernéticos no 2º trimestre, Redação da Abranet, 6 ago. 2019. Disponível em: http://www.abranet.org.br/Noticias/Brasil-sofreu-15-bilhoes-de-tentativas-de-ataques-ciberneticos-no-2%BAtrimestre-2497.html?UserActiveTemplate=site. Acesso em: 7 nov. 2019.

[2] 41,6 bilhões de dispositivos de IoT gerarão 79,4 zettabytes de dados em 2025, 26 jul. 2019. Disponível em: https://seginfo.com.br/2019/06/26/416-bilhoes-de-dispositivos-de-iot-gerarao-794-zettabytes-de-dados-em-2025/. Acesso em: 7 nov. 2019.

[3] What is the Mirai Botnet?. [S. l.]. Disponível em: https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/. Acesso em: 13 set. 2020.

[4] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in Proc. 6th ACM SIGCOMM Conference on Internet Measurement (IMC'06), 2006, pp.41–52.

[5] ROUSE, Margaret. Botnet. [S. l.], 2019. Disponível em: https://searchsecurity.techtarget.com/definition/botnet#:~:text=This%20is%20a%20diagram%20of%20the%20botnet%20command%20and%20control. Acesso em: 21 ago. 2020.

[6] Khan, M. N. A., Chatwin, C. R., and Young, R. 2007b. Extracting evidence from file system activity using bayesian networks. International Journal of Forensic computer science 1, 50– 63.

[7] IMPERVA Website, 2020. Disponível em: https://www.imperva.com/products/ddos-protection-services/. Acesso em: 21 ago. 2020.

[8] Shi, Leyi & Li, Yang & Feng, Haijie. (2018). Performance Analysis of Honeypot with Petri Nets. Information. 9. 245. 10.3390/info9100245

[9] H. R. Zeidanloo, M. J. Z. Shooshtari, P. V. Amoli, M. Safari, and M. Zamani, "A taxonomy of botnet detection techniques," in Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on, vol. 2. IEEE, 2010, pp. 158–162.

[10] URASOV, Stanislav. Creating and Deploying Honeypots in Kubernetes. [S. l.], 13 jun. 2019. Disponível em: https://www.apriorit.com/dev-blog/619-web-cybersecurity-honeypots-in-kubernetes. Acesso em: 21 ago. 2020.

[11] SolarWinds Worldwide. IDS vs. IPS: What's the Difference?. [S. l.], 28 jun. 2019. Disponível em: https://www.dnsstuff.com/ids-vs-ips. Acesso em: 21 ago. 2020.

[12] Honda, Hugo & Facure Matheus & Yaohao, Peng. Os Três Tipos de Aprendizado de Máquina. [S. l.], 27 jul. 2017. Disponível em: https://lamfo-unb.github.io/2017/07/27/tres-tipos-am/. Acesso em: 24 ago. 2020.

[13] SIDHU, Rishi. Use of Cross Validation in Machine Learning. [S. l.], 5 jul. 2019. Disponível em: https://medium.com/x8-the-ai-community/use-of-cross-validation-in-machine-learning-f3b80ad813e6. Acesso em: 21 ago. 2020.

[14] Ashfaque, Johar & Iqbal, Amer. (2019). Introduction to Support Vector Machines and Kernel Methods.

[15] S. Haykin. "Neural Networks: a comprehensive foundation", Prentice Hall, Upper Saddle River, EUA, 1999.

[16] SUPERDATASCIENCE TEAM. Self Organizing Maps (SOM's) - How do Self-Organizing Maps Work?. [S. l.], 28 set. 2018. Disponível em: https://www.superdatascience.com/blogs/self-organizing-maps-soms-how-do-self-organizing-maps-work. Acesso em: 21 ago. 2020.

[17] R, Vinayakumar & Alazab, Mamoun & Kp, Soman & Poornachandran, Prabaharan & Al-Nemrat, A. & Venkatraman, Sitalakshmi. (2019). Deep Learning Approach for Intelligent Intrusion Detection System. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2895334.

[18] FILHO, Mario. As Métricas Mais Populares para Avaliar Modelos de Machine Learning. [S. l.]. Disponível em: https://www.mariofilho.com/as-metricas-mais-populares-para-avaliar-modelos-de-machine-learning/. Acesso em: 18 ago. 2020.

[19] Amazon Web Services, Inc. ou suas afiliadas. Ajuste do modelo: subajuste versus sobreajuste. [S. l.]. Disponível em: https://docs.aws.amazon.com/pt_br/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html. Acesso em: 21 ago. 2020.

[20] Pektaş, Abdurrahman & Acarman, T. (2017). Effective Feature Selection For Botnet Detection Based On Network Flow Analysis.

[21] Sperotto A, Sadre R, Vliet FV, Pras A. A labeled data set for flow-based intrusion detection. In: IPOM '09 Proceedings of the 9th IEEE international Workshop on IP Operations and Management; 2009. pp. 39e50.

[22] Chanthakoummane Y, Saiyod S, Benjamas, N Khamphakdee N. Evaluation Snort-IDS rules for botnets detection; 2016. http://www.it.kmitl.ac.th/ natapon/ncit2015/papers/p87-chanthakoummane.pdf.

[23] Wang, J.; Paschalidis, I.C. Botnet detection based on anomaly and community detection. IEEE Trans. ControlNetw. Syst. 2017,4, 392–404.

[24] STRATOSPHERE LAB. An Empirical Comparison of Botnet Detection Methods. [S. l.], 1 maio 2014. Disponível em: https://www.stratosphereips.org/publications/2014/5/11/an-empirical-comparison-of-botnet-detection-methods. Acesso em: 21 ago. 2020.

[25] GARCÍA , Sebastián. Malware Capture Facility Project. [S. l.]. Disponível em: https://mcfp.weebly.com/about.html. Acesso em: 18 ago. 2020.

[26] Sebastian Garcia, Martin Grill, Honza Stiborek and Alejandro Zunino. An empirical comparison of botnet detection methods". Computers and Security Journal, Elsevier. 2014. Vol 45, pp 100-123. http://dx.doi.org/10.1016/j.cose.2014.05.011

[27] Canadian Institute for Cybersecurity website. CSE-CIC-IDS2018 on AWS. [S. l.], [2018]. Disponível em: https://www.unb.ca/cic/datasets/ids-2018.html. Acesso em: 24 ago. 2020.

[28] V. Kanimozhi and T. P. Jacob, "Artificial intelligence based network intrusion detection with hyper-parameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing"in 2019 International Conference on Communication and Signal Processing (ICCSP), Apr. 2019, pp. 0033–0036.

[29] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”, 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018

[30] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy", IEEE 53rd International Carnahan Conference on Security Technology, Chennai, India, 2019

[31] Sean Miller and Curtis Busby-Earle. “The Role of Machine Learning in Botnet Detection. Conference Proceedings.” December 2016

[32] Stevanovic, Matija, and Jens Myrup Pedersen. ”Machine learning for identifying botnet network traffic.” (2013).

[33] Habibi Lashkari, Arash. (2018). CICFlowmeter-V4.0 (formerly known as ISCXFlowMeter) is a network traffic Bi-flow generator and analyser for anomaly detection. https://github.com/ISCX/CICFlowMeter. 10.13140/RG.2.2.13827.20003.

[34] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley, "Detecting botnets with tight command and control," in Local Computer Networks, Proceedings 2006 31st IEEE Conference on. IEEE, 2006, pp. 195–202.

[35] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in Botnet Detection. Springer, 2008, pp. 1–24.

[36] Beigi, Elaheh & Jazi, Hossein & Stakhanova, Natalia & Ghorbani, Ali. (2014). Towards effective feature selection in machine learning-based botnet detection approaches. 2014 IEEE Conference on Communications and Network Security, CNS 2014. 247-255. 10.1109/CNS.2014.6997492.

[37] Dicas para a configuração de redes neurais. [S. l.], [20?]. Disponível em: http://www.nce.ufrj.br/labic/downloads/dicas_cfg_rna.pdf. Acesso em: 25 ago. 2020.

[38] PANDAS. Pandas.DataFrame.corrwith. [S. l.]. Disponível em: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corrwith.html. Acesso em: 18 ago. 2020.

[39] COH7EIQU8THABU. Ares. [S. l.], 2017. Disponível em: https://github.com/coh7eiqu8thaBu/Ares. Acesso em: 24 ago. 2020.

[40] SANZ, Igor Jochem; LOPEZ, Martin Andreoni. Um Sistema de Detecção de Ameaças Distribuídas de Rede baseado em Aprendizagem por Grafos. In: Anais Principais Do Simpósio Brasileiro de Redes de Computadores e Sistemas Distríbuidos (SBRC), 36. , 2018, Campos do Jordão. Anais Principais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Porto Alegre: Sociedade Brasileira de Computação, may 2018 . ISSN 2177-9384.

[41] Chowdhury, Sudipta & Khanzadeh, Mojtaba & Akula, Ravi & Zhang, Fangyan & Zhang, Song & Medal, Hugh & Marufuzzaman, Mohammad & Bian, Linkan. (2017). Botnet detection using graph-based feature clustering. Journal of Big Data. 4. 14. 10.1186/s40537-017-0074-7.

[42] Tiago P. Peixoto, "The graph-tool python library", figshare. (2014) DOI: 10.6084/m9.figshare.1164194 [sci-hub, @tor]

[43] 5.2.2. Betweenness Centrality. [S. l.]. Disponível em: https://neo4j.com/docs/graph-data-science/current/algorithms/betweenness-centrality/. Acesso em: 18 ago. 2020.

[44] SHAW, Alan. Understanding The Concepts of Eigenvector Centrality And Pagerank. [S. l.], 13 jul. 2019. Disponível em: https://www.strategic-planet.com/2019/07/understanding-the-concepts-of-eigenvector-centrality-and-pagerank/#:~:text=Eigenvector%20centrality%20is%20used%20to,will%%20to%20other%20nodes. Acesso em: 18 ago. 2020.

[45] VETTIGLI, Giuseppe. Minimalistic and NumPy-based implementation of the Self Organizing Map. [S. l.]. Disponível em: https://github.com/JustGlowing/minisom/. Acesso em: 18 ago. 2020.

[46] GOLDSPARROW. Nemty Ransomware. [S. l.]. Disponível em: https://www.enigmasoftware.com/pt/nemtyransomware-remocao/. Acesso em: 18 ago. 2020.

[47] 9 of History's Notable Botnets. [S. l.], [20?]. Disponível em: https://www.whiteops.com/blog/9-of-the-most-notable-botnets. Acesso em: 1 nov. 2019.

[48] Top 10 Botnet Threats in the United States. [S. l.], [20?]. Disponível em: https://www.enigmasoftware.com/top-10-botnet-threats-in-the-united-states/. Acesso em: 1 nov. 2019.

[49] Arshad, Sajjad & Abbaspour, Maghsoud & Kharrazi, Mehdi & Sanatkar, Hooman. (2011). An Anomaly-based Botnet Detection Approach for Identifying Stealthy Botnets. 10.1109/ICCAIE.2011.6162198.