FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Segurança da Informação

Luis Fernando Zanuncio

MACHINE LEARNING PARA DETECÇÃO DE ATAQUE DE NEGAÇÃO DE SERVIÇO

Americana, SP

2018

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Segurança da Informação

Luis Fernando Zanuncio

MACHINE LEARNING PARA DETECÇÃO DE ATAQUE DE NEGAÇÃO DE SERVIÇO

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Segurança da Informação, sob a orientação do (a) Prof.(a) Me. Rossano Pablo Pinto.

Área de concentração: Segurança da Informação.

Americana, SP.

2018

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

Z36m ZANUNCIO, Luis Fernando

Machine Learning para detecção de ataque de negação de serviço. / Luis Fernando Zanuncio. – Americana, 2018.

46f.

Monografia (Curso de Tecnologia em Segurança da Informação) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Rossano Pablo Pinto

1. Inteligência artificial 2. Aprendizado de máquina 3. Segurança em sistemas de informação I. PINTO, Rossano Pablo II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 007.52

681.518.5

Luis Fernando Zanuncio

MACHINE LEARNING PARA DETECÇÃO DE ATAQUE DE NEGAÇÃO DE SERVIÇO

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Segurança da Informação pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Segurança da Informação.

Americana, 05 de dezembro de 2018.

Banca Examinadora:

Rossano Pablo Pinto (Presidente)
Mestre
Fatec Americana

Murilo Fujita (Membro)
Mestre
Fatec Americana

Adriano Cilhos Doimo (Membro)
Especialista
Fatec Americana

AGRADECIMENTOS

Primeiramente, agradeço à famílIa por ser a base da minha educação e dos meus valores, em especial minha mãe Marcia e irmã Priscila.

Em segundo lugar, aos amigos pelo apoio e incentivo à educação, em especial Lucas da Silva Costa pelo apoio neste trabalho.

Por fim, aos docentes e colaboradores da Fatec pelos serviços prestados ao longo de toda a graduação, com ênfase para meu orientador Rossano Pablo Pinto, que foi além do seu dever para me apoiar neste trabalho.

DEDICATÓRIA

Ao meu pai, Elci Zanuncio, que transmitiu a importância da educação em nosso meio familiar, mesmo falecendo antes de poder se graduar ou ver o resultado de seu esforço.

RESUMO

Este trabalho apresenta a aplicação de um algoritmo de *Machine Learning* para detecção de ataque de Negação de Serviço do tipo SYN *Flood*, realizado em ambiente controlado. Tem como objetivo testar e validar a técnica citada, com um alto percentual de acerto, podendo ser utilizada no cenário real das organizações caso adaptada, contribuindo com a Segurança da Informação das mesmas. Por fim, possui o intuito de explorar outras contribuições possíveis a partir dos estudos realizados.

Palavras Chave: *Machine Learning*; ataque de Negação de Serviço; detecção de ataque de Negação de Serviço.

ABSTRACT

This work presents the application of a Machine Learning algorithm for detection of SYN Flood Denial of Service attack, executed in a controlled environment. It aims at testing and validating the mentioned technique, with a high percentage of correctness and it can be used in the real scenario of organizations, if adapted, contributing to their Information Security. Finally, it intends to explore other possible contributions from the studies carried out.

Keywords: *Machine Learning; Denial of Service attack; Denial of Service attack detection.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

DoS – *Denial of Service* (Negação de Serviço);

DDoS - *Distributed Denial of Service* (Negação de Serviço Distribuído);

ML – *Machine Learning;*

kNN - *k-Nearest Neighbors* (k-Vizinhos próximos);

EDA – *Exploratory Data Analysis* (Analise de Dados Exploratória);

IDS - *Intrusion Detection System* (Sistema de detecção de intrusos);

IPS – *Intrution Protection System* (Sistema de proteção de intrusos);

SO – Sistema Operacional;

IP – *Internet Protocol* (Protocolo da Internet).

1 INTRODUÇÃO

A sociedade contemporânea é caracterizada pela tecnologia e conectividade, que está presente de maneira marcante na vida da maioria das pessoas. Essa tecnologia se manifesta em relógios, celulares, *tablets* e outros equipamentos que são utilizados com *softwares*, sistemas e aplicativos.

Na atualidade, onde a conectividade é uma característica evidente da época, há pessoas que se aproveitam para tentar, de alguma maneira, explorar vulnerabilidades com o intuito de alcançar determinados objetivos ilícitos. As pessoas e as organizações precisam se prevenir.

No caso das organizações, o cenário é mais rigoroso. Todos os ativos de informação possuem um determinado grau de significância, devendo ser protegidos.

No caso da rede da empresa estar conectada à internet, é necessário tomar medidas específicas para prevenir que agentes mal intencionados não causem danos. São algumas medidas:

> O roteador que conecta a rede da empresa à internet deve ser configurado corretamente. [...]
>
> Cada camada na rede da empresa deve ser protegida por um *firewall*, com as regras de segurança aplicadas pelo *firewall* ficando cada vez mais rígidas com cada camada. [...] A finalidade do *firewall* é proteger e evitar o acesso não-autorizado ao segmento de rede que ele protege. [...] alguns métodos que o *firewall* pode usar:
>
> Filtragem de pacotes [...]
>
> *Gateway* de aplicativo [...]
>
> *Gateway* de nível de circuito [...]
>
> Servidor proxy [...]. (OPPEL, 2004, p. 193-195).

Machine Learning é uma tecnologia conhecida há décadas, mas que vem sendo aplicada no cenário corporativo gradativamente. A motivação para abordar o tema pode ser dividida em duas modalidades, sendo elas:

a) Acadêmico: o estudo do tema pode apontar novas estratégias para a segurança da informação;

b) Social: o desenvolvimento do tema abre a oportunidade para que pessoas da área de segurança da informação adotem novas metodologias para proteção de um determinado produto ou dado.

O projeto acadêmico científico é norteado por objetivos, sendo estes verificar se a utilização da tecnologia do *Machine Learning* para detecção de ataque de negação de serviço é viável. Para tal, se faz necessário uma alta taxa de acerto na classificação dos pacotes maliciosos e de fluxo normal durante um ataque de negação de serviço SYN *Flood*, através do algoritmo kNN de *Machine Learning* em Python, de modo que possa detectar com precisão essa situação.

Este trabalho está estruturado como se segue: este capítulo apresenta a introdução. O segundo capítulo desenvolve toda a fundamentação teórica referente ao tema central e demais elementos fundamentais, como segurança da informação e redes de computadores. No terceiro capítulo aprofunda-se quanto ao funcionamento do algoritmo de *Machine Learning* adotado. O quarto capítulo apresenta o método e técnica utilizado para a elaboração do estudo e experimento, bem como detalhes do cenário. O quinto capítulo apresenta os detalhes da implementação e a execução do experimento. Por fim, nas Considerações Finais, demonstrou-se as principais informações almejadas e obtidas graças ao conteúdo estudado.

2 FUNDAMENTAÇÃO TEÓRICA

Tanto Segurança da Informação quanto Redes de Computadores são áreas do saber diretamente relacionados ao referido estudo. Para tal, se faz necessário a definição de alguns conceitos essenciais para a correta compreensão do tema.

Fontes (2010, p. 6) cita em sua obra que a informação sempre foi um dos temas mais importantes da organização. Desde que passou a ser digitalizada, sua proteção precisa ser levada a sério pelos empresários e proprietários da organização.

A Segurança da Informação é fundamentada em três pilares. Elmasri e Navathe (2011, p. 2-3) dizem que os pilares, sendo eles a integridade, confidencialidade e disponibilidade, precisam ser protegidos.

a) a integridade refere-se à proteção contra modificação imprópria. A modificação pode ser efetuada de maneira intencional ou não, por vezes sendo uma falha do próprio sistema. Sua perda compromete a garantia da informação armazenada;

b) confidencialidade, por sua vez, refere-se à proteção contra exposição não autorizada, o que envolve questões legais, como o *Data Privacy Act*. Exemplo é o acesso de um funcionário aos dados pessoais de outro funcionário, tais como endereço, documentos, de modo que fuja das políticas da empresa e das leis;

c) a disponibilidade, por último, refere-se a garantir que determinado ativo possa ser acessado quando necessário pelos usuários permitidos. Essa ameaça é a principal para o referido projeto, sendo tratada de maneira mais completa nos próximos capítulos.

A Segurança da Informação se torna cada vez mais impactante na vida dos usuários. O progresso tecnológico dos processadores e meios de comunicação permitiram diversos avanços e do mesmo modo grandes riscos às organizações.

2.1 Rede de computadores

Redes de computadores, para Tanembaum (2003, p. 18) é um conjunto de computadores autônomos interconectados. Dois computadores estão interconectados quando conseguem trocar informações.

A Internet é definida por Kurose (2010, p. 2) como uma rede de computadores – *laptops*, celulares, *webcams*, etc. - que interconecta milhares de dispositivos computacionais em todo o planeta. Os dados são segmentados em pacotes e enviados, através de enlaces (*links*) e comutadores, do emissor para o receptor. Esses pacotes podem trafegar por diferentes rotas, mas a informação é reunida quando chega ao destino.

Protocolos são elementos essenciais para que ocorra a comunicação. Para Tanembaum (2003, p. 37), coletivamente, as regras e convenções usadas nesse diálogo são conhecidas como o protocolo da camada *n*. Basicamente, um protocolo é um acordo entre as partes que se comunicam, estabelecendo como se dará a comunicação.

Todos os ativos envolvidos na rede executam protocolos que controlam desde o envio até o recebimento de informações, conforme explica Kurose (2010, p. 4-5). Todas as atividades na internet que envolvem duas ou mais entidades que se comunicam são geridas através de protocolos.

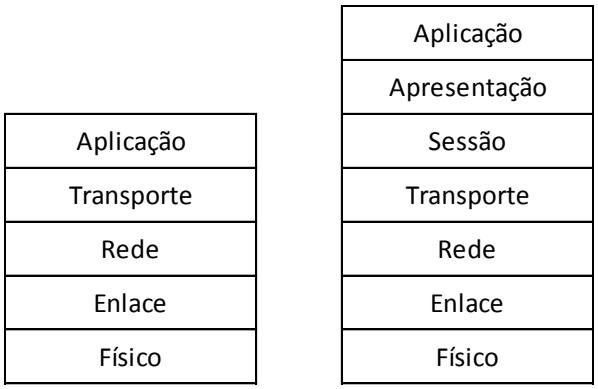
“O TCP [...] e o IP [...] são dois dos protocolos mais importantes da Internet. O protocolo IP especifica o formato dos pacotes que são enviados e recebidos entre roteadores e sistemas finais” (KUROSE, 2010, p. 4).

Antes de aprofundar nos protocolos, é necessário trazer o conceito das camadas, que são necessárias para que ocorra qualquer tipo de comunicação entre dispositivos.

“Uma arquitetura de camadas nos permite discutir uma parcela específica e bem definida de um sistema grande e complexo. [...] Provê modularidade fazendo com que fique muito mais fácil modificar a implementar” (KUROSE, 2010, p. 36)

A Figura 1 apresenta o comparativo entre dois modelos diferentes de classificação das camadas, sobre as quais são construídos os protocolos. Há ainda uma terceira representação, com 4 camadas, onde soma-se a camada física de enlace do primeiro modelo a seguir.

Figura 1 – A pilha de protocolo da Internet e o modelo de referência OSI



a. Pilha de protocolo da Internet de cinco camadas

b. Modelo de referência ISO OSI de sete camadas

Fonte: Kurose, 2010, p. 38

A camada que é aprofundada neste documento é a de Transporte, sobre a qual é realizada o ataque de negação de serviço, conforme explicado na seção 2.1.1.

2.1.1 Camada de transporte

A camada de transporte é essencial para o funcionamento de uma rede de computadores. Segundo Forouzan (2010, p. 37), ela é a responsável pela entrega processo a processo, garantindo que a mensagem chegue intacta e na sequência correta, supervisionando tanto o controle de erros como o controle de fluxo. Ela também é responsável por:

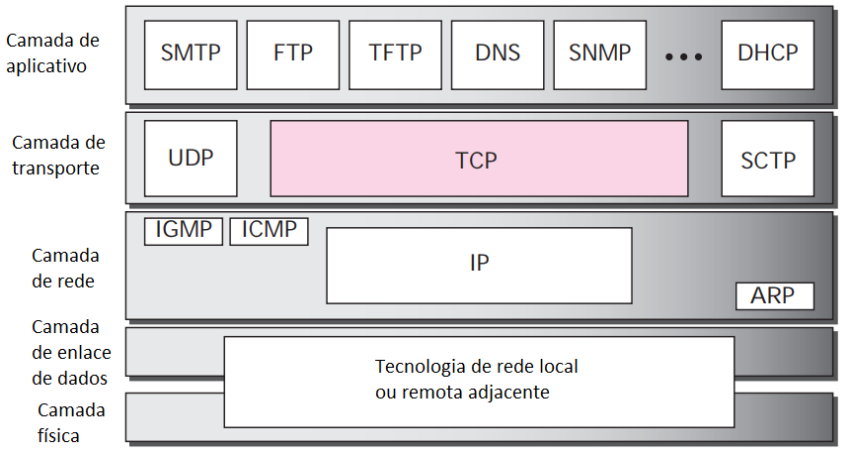
a) Realizar o endereçamento do ponto de acesso ao serviço;
b) Segmentação e remontagem de pacotes;

c) Controle de conexão;
d) Controle de fluxo; e
e) Controle de erros.

> A finalidade dessa camada é permitir que as entidades pares dos *hosts* de origem e de destino mantenham uma conversação[..]. O TCP (Transmission *Control Protocol* — protocolo de controle de transmissão), é um protocolo orientado a conexões confiável que permite a entrega sem erros de um fluxo de bytes originário de uma determinada máquina em qualquer computador da inter-rede. Esse protocolo fragmenta o fluxo de bytes de entrada em mensagens discretas e passa cada uma delas para a camada inter-redes. No destino, o processo TCP receptor volta a montar as mensagens recebidas no fluxo de saída. O TCP também cuida do controle de fluxo, impedindo que um transmissor rápido sobrecarregue um receptor lento com um volume de mensagens maior do que ele pode manipular. (TANENBAUM, 2003, p. 49)

A Figura 2 apresenta as camadas bem como alguns dos protocolos mais utilizados em cada uma.

Figura 2 - Protocolos e redes no modelo TCP/IP



Fonte: Forouzan, 2008, p. 275

Forouzan (2008, p. 275) especifica que o conjunto TCP/IP possui dois protocolos nessa camada: o UDP e o TCP. A segunda é explicada detalhadamente na seção 2.1.2, pois foi empregada no experimento.

2.1.2 Protocolo TCP

Forouzan (2008, p. 6) define protocolo como um conjunto de regras, sendo divididos em:

a) Sintaxe – Estrutura ou formato dos dados;
b) Semântica – Qual o conteúdo de cada sessão de bits; e
c) Sincronismo – Quando e com qual rapidez os dados devem ser enviados.

Da mesma maneira que o UDP, o TCP é um protocolo de camada de transporte que funciona de processo para processo, levando em consideração o número de porta, conforme Fourozan (2008, p. 276). Não obstante, o TCP funciona orientado a conexão, criando uma conexão entre emissor e receptor para envio dos dados.

O protocolo TCP permite o envio dos dados por meio de fluxo, como comenta Forouzan (2008, p. 227-229), como se criasse um “tubo” imaginário. Como o transmissor e receptor podem possuir diferentes capacidades de processamento, para que nenhum dado seja perdido, se faz necessário o uso de *buffers* de armazenamento, que acabam por serem utilizados para controle de fluxo e erros. O receptor ordena os pacotes para serem lidos em sequência utilizando esse espaço. Após serem lidos, a área é liberada permitindo que outros pacotes possam ser armazenados temporariamente. Diferentes processos podem se comunicar de maneira segura, mesmo com diferentes áreas de *buffers,* velocidades de transmissão e tamanho dos pacotes. Quando um processo A (emissor) envia dados para outro processo B (receptor) ocorre:

1. Os dois TCPs estabelecem a conexão entre eles.
2. Dados são trocados nas duas direções enquanto necessário.
3. A conexão é encerrada ao término.

O segmento TCP, conforme explica Forouzan (2008, p. 282), possui um cabeçalho fixo de 20 bytes. Diversos campos compõem este, entre eles o de controle. Aqui, são definidas 6 *flags*. São elas que permitem o funcionamento do

protocolo, tendo papel no controle do fluxo, estabelecimento da conexão, término da conexão, entre outros. A Tabela 1 apresenta a descrição de cada uma.

Tabela 1 – Descrição das Flags no campo controle

Flag	*Desrição*
URG	O valor do campo de ponteiro urgente é válido.
ACK	O valor do campo de reconhecimento é válido.
PSH	Acrescenta os dados.
RST	A conexão deve ser redefinida.
SYN	Sincroniza os números de sequência durante a conexão.
FIN	Termina a conexão.

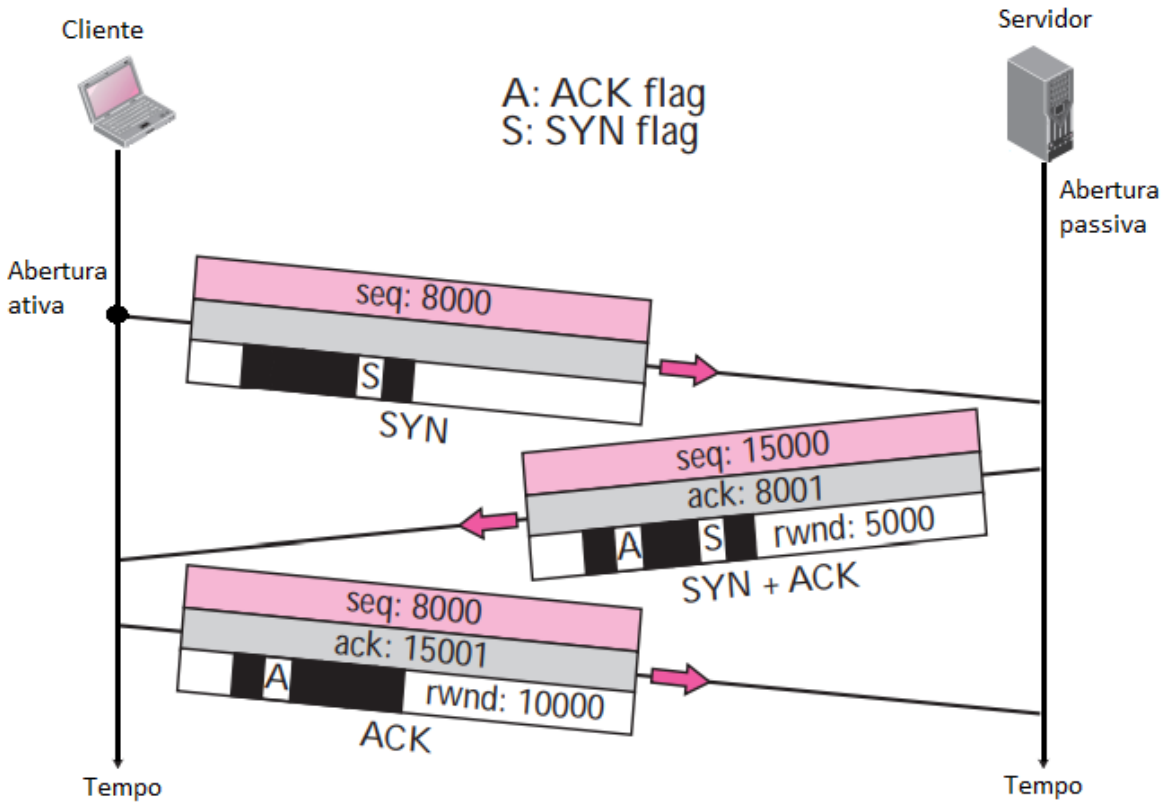
Fonte: Forouzan, 2008, p. 283

Para que dois equipamentos possam estabelecer uma conexão TCP, é necessário que ocorra o reconhecimento recíproco de três vias, conforme Forouzan (2008, p. 285). O primeiro passo é do servidor, que precisa estar aberto para receber conexões, denominado abertura passiva. Após isso, é necessário que um processo cliente faça um pedido de abertura ativa ao servidor. Feito isso, o TCP inicia o processo de estabelecimento de conexão, que é descrito nos passos abaixo:

1. O cliente envia um primeiro segmento, com *flag* SYN, que é necessário para sincronizar o número de sequência, gerado aleatoriamente nessa primeira etapa.
2. O servidor responde com o segundo seguimento, contendo as *flags* SYN + ACK. O número de sequência é inicializado no servidor e o cliente é informado de qual o próximo número de sequência que espera receber do servidor.
3. Por fim, o cliente responde com a *flag* ACK, confirmando a conexão com o servidor. Nesse ponto os dados podem começar a ser transmitidos.

O processo citado acima, também conhecido como 3-*way handshake*, existe sempre no início de uma comunicação. A Figura 3 ilustra o processo.

Figura 3 – Estabelecimento de Conexão TCP



Fonte: Forouzan, 2008, p. 286

As camadas e os protocolos apresentam modos de operar distintos e, do mesmo modo, vulnerabilidades diversas. Da mesma forma, podem ser exploradas de maneiras diferentes para realizar ataques digitais.

2.2 Ataques digitais

Assim como há diversos crimes que ocorrem no mundo físico, o mesmo se aplica à realidade da computação. Diariamente há ataques e fraudes cibernéticas.

Segundo o CERT.br, qualquer serviço, computador ou rede que seja acessível via Internet pode ser alvo de um ataque, assim como qualquer computador com acesso à Internet pode participar de um ataque.

Segundo Kurose (2010, p. 41), a internet se tornou essencial, desde usuários domésticos, até grandes empresas e organizações, sendo todos sujeitos a

indivíduos mal-intencionados, que buscam danificar equipamentos, roubar informações ou tornar os serviços inoperantes.

O CERT.br também menciona sobre a Segurança em Redes, onde o grande aumento da velocidade de conexão e o aumento do número de dispositivos são agravantes nesse cenário, pois há mais maneiras de serem explorados.

Existem diversas técnicas, ferramentas e formas de se realizar um ataque. Um dos conceitos mais conhecidos são vírus, que contaminam a máquina do hospedeiro e se espalha naquele equipamento. Atualmente há uma crescente utilização dos *Ransomwares*, que contaminam uma ou mais máquinas e criptografam seus dados, exigindo pagamento para fornecer a chave que traria os dados de volta, porém sem garantias.

Conforme Stallings (2008, p. 5-6) os ataques que utilizam a internet evoluíram, além de que os requisitos, seja conhecimento ou tecnologia, para realizá-los, diminuiu. Além disso, passaram a ser automatizados e causam danos maiores. Ele relaciona esses pontos com a crescente utilização da internet. Por fim, define ataque à segurança como qualquer ação deliberada, derivada de uma ameaça inteligente, que compromete a segurança da informação pertencente a uma organização.

Um ataque de rede, que pode ocorrer na rede interna ou pela internet, é o ataque de negação de serviço, ou *Denial of Service* (DoS) em inglês. Este tipo de ataque é classificado por Stallings (2008, p. 7) como sendo do tipo ativo. Ele impede ou inibe o uso ou gerenciamento normal de uma aplicação.

2.2.1 Ataque de Negação de Serviço

Segundo Kurose (2010, p. 42), o ataque DoS torna a rede, o hospedeiro ou parte da infraestrutura inutilizável por usuários comuns, afetando a disponibilidade do serviço, recurso ou ferramenta. Além disso, ele cita que são extremamente comuns, podendo ser divididos em três categorias, assim como o CERT.br:

a) Ataque de vulnerabilidade: pacotes normais são enviados a uma aplicação vulnerável ou sistema operacional sendo executado em um hospedeiro direcionado, explorando-o.

b) Inundação na largura de banda: São enviados uma grande quantidade de pacotes, que acabam por atingir os limites físicos de comunicação do meio.
c) Inundação na conexão: O atacante estabelece um grande número de conexões TCP (semiabertas ou abertas) na vítima. As requisições falsas fazem com que conexões legítimas sejam perdidas.

O ataque que pretende-se detectar neste trabalho pertence à terceira categoria citada acima. O ataque de negação de serviço é caracterizado por prejudicar apenas a disponibilidade de um determinado ativo ou serviço.

> Negação de serviço, ou DoS (*Denial of Service*) é uma técnica pela qual um atacante utiliza um computador para tirar de operação um serviço, um computador ou uma rede conectada à Internet. Quando utilizada de forma coordenada e distribuída, ou seja, quando um conjunto de computadores é utilizado no ataque, recebe o nome de negação de serviço distribuído, ou DDoS (*Distributed Denial of Service*). (CERT.br, 2012, p. 21).

Para Forouzan (2008, p. 287) o TCP está suscetível à ataque de inundação SYN. Ele explica que o atacante mascara o endereço do emissor e envia uma grande quantidade de solicitações para estabelecimento de conexão (*flag* SYN). Isso ocupa recursos do servidor, que pode levá-lo a falhar. Ele comenta de algumas estratégias tradicionais para evitar esse problema, como limite de pedidos de conexão durante um tempo e filtragem de pacotes de acordo com a origem. Mas em ambos os casos, clientes legítimos podem ser prejudicados pelas táticas.

Para Larson e Stephens (2000, p. 321) o ataque adotado no experimento apresentado neste trabalho é um tipo de ataque de negação de serviço denominado Ataque de Inundação SYN. O número de conexões TCP que um dispositivo é capaz de ter é limitado. Devido as requisições maliciosas SYN, o servidor perderá sua capacidade de realizar a abertura de novas conexões (*Three-way handshake)* em algum momento, ocorrendo a negação de serviço.

2.2.2 Detecção de ataque

Mesmo um ambiente bem projetado precisa ser monitorado. Para Lyra (2008, p. 101), todo o comportamento anormal de um sistema deve ser examinado. Somente assim determinado comportamento pode ser considerado ou não um incidente de segurança da informação.

Para Kurose (2010, p. 540) mesmo recursos de segurança como *firewall* e *gateway* não são suficientes para detectar diferentes tipos de ataque. Para isso se faz necessário a inspeção profunda de pacote. Um dos mecanismos capazes de fazer essa função é o IDS.

> Os sistemas de detecção de intrusão, ou simplesmente IDS (*Intrusion Detection System*) são aplicações que monitoram e correlacionam uma série de eventos, que ocorrem em uma rede e em sistemas computacionais, em geral. [...] O IDS examina os *logs* procurando por eventos, ou por uma combinação deles, que possam indicar um comportamento suspeito na rede ou em uma máquina. [...] Os sistemas de IDS, em função dos tipos de eventos que registram e de sua localização em um sistema computacional, são classificados como: baseado em hospedeiro (*host*) e baseado em rede. [...] O segundo tipo de IDS, baseado em rede, monitora o tráfego de uma rede em busca de padrões fora do comum que poderiam indicar ataques, como, por exemplo, de negação de serviço através de SYN *flooding*. [...] Um dos problemas para se utilizar um IDS é configurá-lo adequadamente. (CARISSIMI, 2008, p. 379)

Outro equipamento que pode ser utilizado é o IPS, sistema de prevenção de intrusos. Diferente do IDS, estes filtram o tráfego suspeito, independente de ação humana. Ele também realiza uma análise profunda dos pacotes para poder classificá-los, conforme comenta Kurose (2010, p. 542-543). Ele explica que esses sistemas são classificados como baseados em assinaturas ou sistemas baseados em anomalias.

Os baseados em assinatura possuem uma extensa base de dados de assinaturas que são informações que caracterizam determinados ataques. Após ler cada pacote, compara a sua base para verificar se é compatível. Não obstante, este modelo possui algumas desvantagens, sendo elas:

a) São falhos quando há novos tipos de ataques ainda não registrados;
b) Falsos positivos;
c) Altos requisitos de processamento para funcionar, deixando passar pacotes.

Já o sistema baseado em anomalias funciona a partir da análise do perfil do tráfego, comparando estatisticamente do momento da captura com o de um funcionamento normal. Mesmo sendo eficiente contra novos tipos de ataques, a implementação é mais complexa, que o torna menos usado.

A atual pesquisa visa considerar uma outra tecnologia, denominada *Machine Learning*, para que possa realizar de maneira eficiente a detecção de um ataque.

Para Doshi (2018, p. 2) as técnicas de detecção de anomalias tradicionais, como IDS, quando aplicadas para detectar tráfego anormal, podem gerar muitos falsos positivos, além de não serem capazes de evoluir junto aos ataques. Esse cenário faz necessário a utilização de técnicas mais sofisticadas, como o *Machine Learning*. Além disso, considera alguns algoritmos promissores, entre eles o kNN.

2.3 Machine Learning

Machine Learning é um conceito que existe há décadas, mas que passou a ser explorado, principalmente no âmbito comercial, mais recentemente. O emprego desta tecnologia pode trazer diversos benefícios à segurança da informação.

> Ele [*Machine Learning*] nasceu do reconhecimento de padrões e da teoria de que máquinas podem aprender sem serem programadas para realizar tarefas específicas; pesquisadores interessados em inteligência artificial queriam saber se computadores podem aprender com dados. O aspecto interativo do *machine learning* é importante porque, conforme os modelos são expostos a novos dados, eles conseguem se adaptar independentemente. Eles aprendem com cálculos anteriores para produzirem decisões e resultados confiáveis, passíveis de repetição. Essa ciência não é nada nova — mas está ganhando um novo impulso. (SAS, 2018)

Segundo Harrington (2012, p. 5), o *Machine Learning* é utilizado em situações onde os dados brutos não são óbvios de serem adivinhados. Ele comenta que o conceito está na intersecção da engenharia, computação e estatística. A estatística é aplicada, pois há diversas situações onde a resolução de um problema não é exata, e sim probabilística.

Segundo a SAS (2018), existem dois métodos mais adotados de *Machine Learning*, sendo eles:

a) Algoritmos de aprendizagem supervisionada - são treinados utilizando exemplos rotulados. O algoritmo de aprendizagem recebe um conjunto de entradas com as saídas classificadas correspondentes e aprende de maneira matemática, ao comparar a saída real com as saídas corretas para encontrar os erros, de acordo com o algoritmo empregado. Ele, então, modifica o modelo de acordo. Através de métodos como classificação, regressão, e aumento de previsões e de gradientes, a aprendizagem supervisionada utiliza funções e padrões para prever os valores de entrada reais.

b) Aprendizagem não-supervisionada - é utilizada com dados que não possuem bases conhecidas. O algoritmo deve descobrir o que está sendo mostrado e classificar. O objetivo é explorar os dados e encontrar alguma estrutura dentro deles que permita sua categorização. Aprendizagem não-supervisionada funciona bem com dados transacionais. Sua implementação é mais complexa.

A Tabela 2 exibe alguns dos principais algoritmos utilizados de *Machine Learning*.

Tabela 2 – Algoritmos de *Machine Learning*

Supervisionado	
k-Nearest Neighbors	Linear
Naive Bayes	Lineal Ponderado localmente
Support vector machines	Pico
Decision tress	LASSO
Não Supervisionado	
k-Means	Maximização de expectativas
DBSCAN	Janela Parzen

Fonte: Autoria própria baseado em Harrington (2012, p. 10)

Para se desenvolver uma aplicação de *Machine Learning*, Harrington (2012, p. 21) define algumas etapas, sendo elas:

a) Coletar os dados;
b) Preparar dados para análise;
c) Analisar os dados;
d) Treinar o algoritmo; e
e) Testar o algoritmo.

Para Harrington (2012, p. 1), a aprendizagem supervisionada é menos complexa, pois acaba sendo resumida em testar o valor de entrada (seja ele nominal ou numérico) para então classificá-lo.

Com base nessas informações, considera-se utilizar o método de aprendizagem supervisionado mais adequado para ser aplicado na situação, junto ao algoritmo kNN. Após treinado, a aplicação foi capaz de distinguir com alto grau de acerto as solicitações legítimas e de ataque do experimento.

3 ALGORITMO KNN EM MACHINE LEARNING

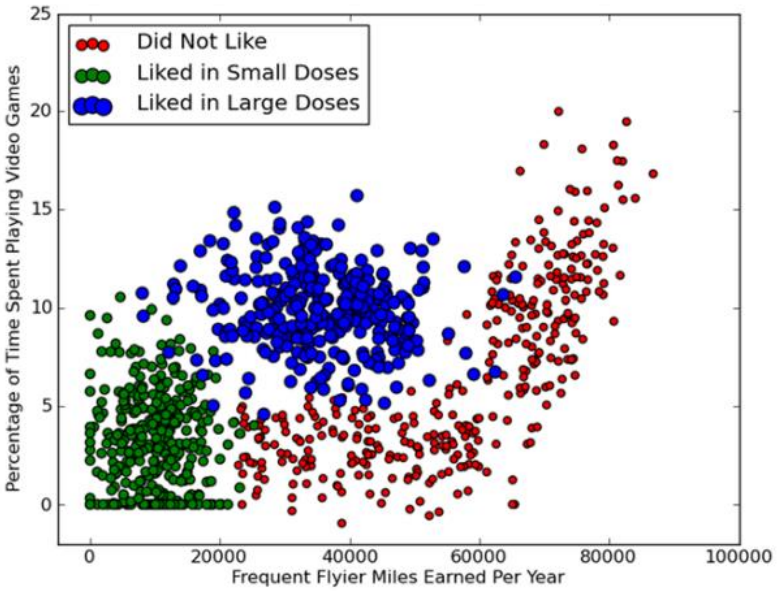
O kNN é a abreviação de um algoritmo de *Machine Learning*, denominado *k-Nearest Neighbors.* Segundo Harrington (2012, p. 19), seus pontos positivos são a alta taxa de acerto e não ser prejudicado por dados fora da normalidade.

Por outro lado, requer grande capacidade computacional, com grandes quantidades de memória para trabalhar com grandes conjuntos de dados.

Harrington (2012, p. 19) explica que no kNN há um conjunto existente de dados de exemplo, denominado conjunto de treinamento. Há rótulos para todos esses dados - sabemos em que classe cada parte dos dados deve se encaixar. Quando se recebe um novo dado sem um rótulo, compara-se esse novo item com todos os dados existentes. Em seguida, é levado em consideração a classificação dos vizinhos mais próximos. É observado os *k* vizinhos mais semelhantes geometricamente de dados do nosso conjunto de treinamento; é daí que vem o *k*, um número inteiro e normalmente é menor que 20. Por último, a maioria dos votos dos k dados mais semelhantes determina a classificação do dado testado.

O Gráfico 1 é um exemplo que apresenta visualmente como os dados são organizados em *clusters*.

Gráfico 1 – Exemplo de dados no kNN



Fonte: Harrington, 2012, p. 29

Observa-se no Gráfico 1 que há três rótulos aos quais os dados foram agrupados, seguindo o exemplo de Harrington. Apresentado dessa forma, é possível constatar a formação de *clusters*, ou núcleos, onde os dados são aglutinados. Ainda, pode-se reparar que em diversos pontos os dados estão dentro da área de outro grupo. Estes são os mais propensos a serem classificados incorretamente.

Para determinar quais são os vizinhos mais próximos, é utilizado a Distância Euclidiana, uma medida de dissimilaridade. Segundo Cunha (2017, p. 10), é obtida através da formula apresentada na Figura 4, revelando a distância entre dois pontos apresentados no plano cartesiano.

Figura 4 – Distância Euclidiana

$$d_{ij} = \left\{ \sum_{k=1}^{p} \left(X_{ik} - X_{jk} \right)^2 \right\}^{1/2}$$

Fonte: Cunha, 2017, p. 10

Ao aplicar o teorema de Pitágoras, a distância entre os pontos passa a poder ser calculada. A fórmula empregada é apresentada na Figura 5.

Figura 5 – Fórmula transcrita

$$d = \sqrt{(xA_0 - xB_0)^2 + (xA_1 - xB_1)^2}$$

Fonte: Harrington, 2012, p. 23

Uma vez que a distância entre todos os *k* vizinhos mais próximos é determinada, o algoritmo analisa qual a classificação da maioria e, com base nisso, qual será o rótulo do dado apresentado.

Segundo Scikit Learn (2018), o princípio por trás dos métodos de vizinho mais próximo é encontrar um número pré-definido de amostras de treinamento mais próximas da distância do novo ponto e prever o rótulo a partir delas. O número de amostras pode ser uma constante definida pelo usuário (k-vizinhos, utilizado no estudo) ou variar com base na densidade local de pontos (aprendizado de vizinho baseado em raio). A distância pode, em geral, ser qualquer medida métrica: a distância Euclidiana padrão é a escolha mais comum. Os métodos baseados em vizinhos são conhecidos como métodos de aprendizado de máquina não generalizantes, já que eles simplesmente “lembram” todos os seus dados de treinamento.

A classificação baseada em vizinhos é um tipo de aprendizado baseado em instância ou de aprendizado não generalizante: ela não tenta construir um modelo interno geral, mas simplesmente armazena instâncias dos dados de treinamento, conforme comenta Scikit Learn (2018). Por conta disso, nesse algoritmo em específico, não há a etapa de treinamento, conforme comenta Harrington (2012, p. 25).

Em sua obra, Harrington ilustra na página 20 um exemplo aplicado e didático para compreensão do kNN. No mesmo intuito, elaborou-se um cenário em que o algoritmo poderia ser aplicado.

Inicialmente, os dados precisam ser coletados e transformados. Na situação exemplo, há 12 dados conhecidos, divididos nas categorias Azul e Vermelho. Além disso, um 13º dado com valores conhecidos para ser categorizado pelo kNN, conforme ilustrado na Tabela 3.

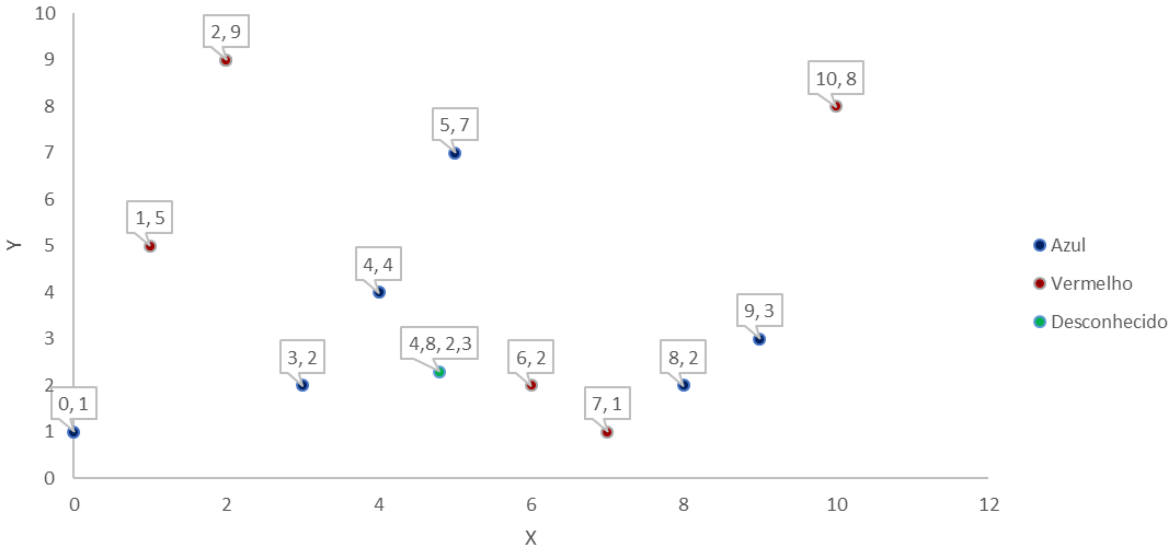
Tabela 3 – Exemplo kNN

Num	X	Y	Categoria
1	4	4	Azul
2	2	9	Vermelho
3	8	2	Azul
4	1	5	Vermelho
5	3	2	Azul
6	2	9	Vermelho
7	5	7	Azul
8	7	1	Vermelho
9	0	1	Azul
10	6	2	Vermelho
11	9	3	Azul
12	10	8	Vermelho
13	4,8	2,3	Desconhecido

Fonte: Elaborado pelo autor

Os dados conhecidos são lidos pelo algoritmo e são armazenados em *arrays* (vetores ou matrizes) na memória. No referido exemplo, os mesmos podem ser representados graficamente através do Gráfico 2.

Gráfico 2 – Gráfico do exemplo



Fonte: Elaborado pelo autor

Quando um dado é levado para ser testado, como o 13º neste exemplo, o kNN irá calcular a distância euclidiana do mesmo em função de todos os demais pontos. A demonstração desse cálculo para o ponto 10 (6,2) e 13 (4,8, 2,3) estão na Figura 6.

Figura 6 – Distância entre os pontos 10 e 13

$$\sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$$

$$\sqrt{(6 - 4{,}8)^2 + (2 - 2{,}3)^2}$$

$$\sqrt{(1{,}2)^2 + (-0{,}3)^2}$$

$$\sqrt{1{,}44 + 0{,}09}$$

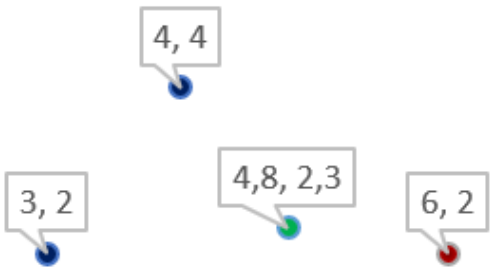
$$\sqrt{1{,}53}$$

$$1{,}23693$$

Fonte: Elaborado pelo autor

Os *k* vizinhos mais próximos, valor inteiro geralmente maior que 3 e menor que 20, serão então utilizados para rotular o dado desconhecido. Na Figura 7considera-se que *k* será igual à 3 (três).

Figura 7 – Vizinhos mais próximos



Fonte: Elaborado pelo autor

Uma vez que a distância dos *k* vizinhos mais próximos é conhecida, vide Tabela 4, o algoritmo leva em consideração qual a maioria dos rótulos para determinar.

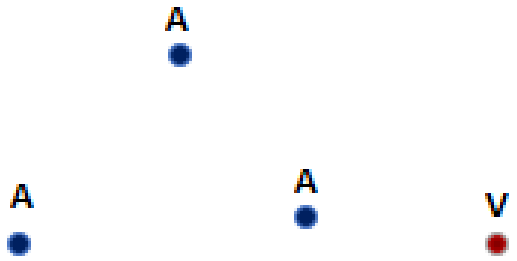
Tabela 4 – Distância Euclidiana

Dado	Distância
1 (Azul)	1,878829
5 (Azul)	1,824829
10 (vermelho)	1,236932

Fonte: Elaborado pelo autor

Apesar do dado 10 (vermelho) estar mais próximo do ponto desconhecido, o mesmo será determinado como azul, pois 1 e 5 possuem este rótulo, conforme Figura 8.

Figura 8 – Classificação kNN



Fonte: Elaborado pelo autor

Para Doshi (2018, p. 5-6), o kNN se mostrou altamente preciso, equiparando-se com bons resultados de acerto, superando outros algoritmos testados no mesmo cenário de negação de serviço. Por fim, a facilidade de implementação foi o último elemento que envolveu a escolha deste algoritmo para ser empregado no estudo.

4 CENÁRIO

O cenário a ser utilizado foi reproduzido em laboratório, porém preserva características do funcionamento normal de uma rede de computadores e de um ataque de negação de serviço.

A metodologia pode ser definida como um conjunto de procedimentos utilizados no trabalho acadêmico. Aplicada ao pensamento científico, segue o método indutivo ou dedutivo, utilizando este último neste projeto, partindo da ideia de que duas preposições formam sempre uma conclusão. “É um conhecimento que se obtém de forma inevitável e sem contraposição. Parte do geral para o particular, do conhecimento universal para o conhecimento particular”. (FACHIN, 2006, p. 32)

4.1 Método de abordagem e procedimento

Para a elaboração dos experimentos, é empregado o Método da Pesquisa de Campo. Também chamada de pesquisa empírica, requer contato maior com a população pesquisada a fim de verificar a ocorrência de algum fenômeno que esteja influenciando ou realizar alguma experiência. No caso em questão, é empregado apenas em forma de experiência.

4.1.1 Técnicas de pesquisa

Entre as técnicas existentes, serão utilizadas as descritas na sequência:

a) Pesquisa bibliográfica – Esse tipo de pesquisa, segundo Marconi e Lakatos (2009, p. 57), abrange toda a bibliografia já tornada pública em relação ao tema estudado. Dentre os tipos de fontes bibliográficas, serão utilizados livros e artigos científicos.

b) Pesquisa tecnológica ou aplicada (prática) – Essa forma de pesquisa, ainda segundo Marconi e Lakatos (2009 p. 69), pode ser aplicada de forma imediata à diversos campos do saber.

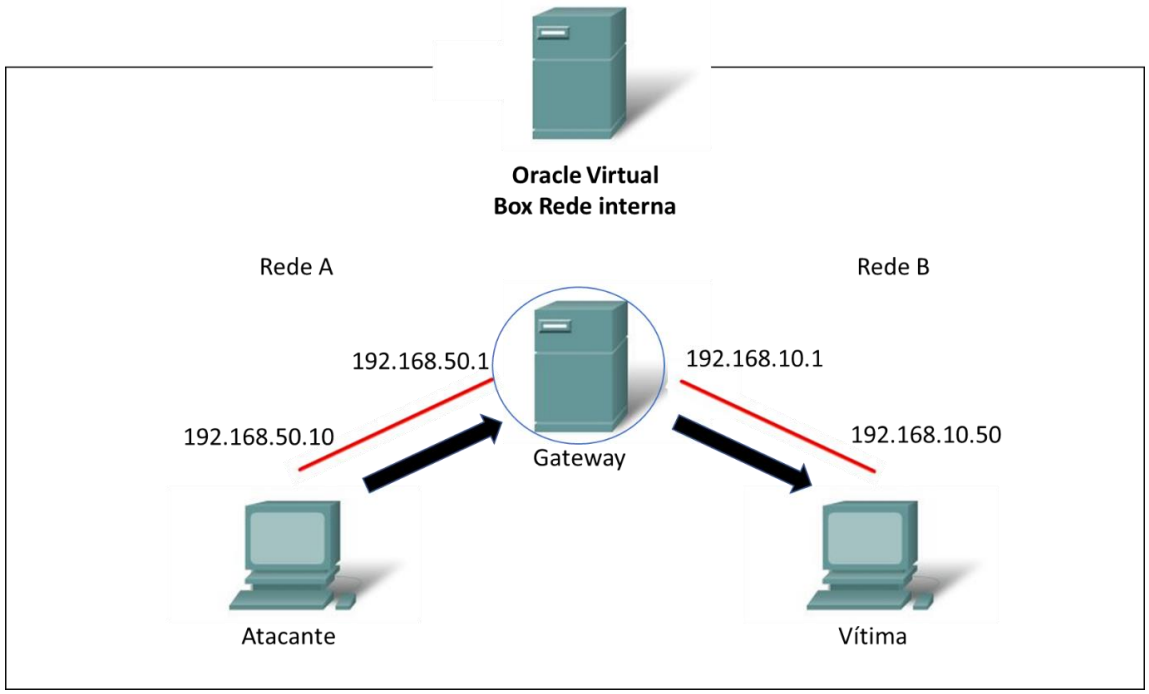
4.3 Realizando a negação de serviço

Para compreender o ataque, denominado SYN *Flood*, foi utilizado um *framework* de *Metasploit* no Kali. Utilizando os comandos do programa, foi definido o *host* como vítima através de seu número IP e iniciado o ataque. O mesmo somente cessou quando o comando foi abortado na máquina atacante.

Para atingir o ponto de negação de serviço na vítima, o ambiente foi preparado com três máquinas virtuais. O atacante, sendo um SO Linux Kali com 1 Gb de memória RAM. A vítima, sendo um SO Linux CentOS com 512 Mb de memória RAM. Por fim, o *Gateway*, com SO Linux Debian com 1024 de memória RAM.

O atacante e o *Gateway* estão conectados por uma rede interna denominada A. Com outra interface de rede, o *Gateway* está conectado através de outra rede interna B com a vítima e é nele que o tráfego será capturado utilizando o analisador de pacotes tcpdump. O IP de origem dos pacotes maliciosos foi definido aleatoriamente pelo programa empregado. O cenário está representado na Figura 9.

Figura 9 – Cenário do ataque



Fonte: elaborado pelo autor

O ataque foi executado até que a vítima tivesse de fato o serviço prejudicado pelo esgotamento dos recursos para atender as requisições. É evidenciado pelo tráfego exibido no Wireshark, conforme Figura 10, onde as novas conexões TCP foram recusadas.

Figura 10 – Conexões recusadas pela vítima

	Time	Source	Destination	Protocol	Length	Info
3449…	471.719771	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 12055 → 80 [SYN] Seq=0
3449…	471.721105	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 56654 → 80 [SYN] Seq=0
3449…	471.722442	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 15304 → 80 [SYN] Seq=0
3449…	471.723846	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 3483 → 80 [SYN] Seq=0 W
3449…	471.725240	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 63062 → 80 [SYN] Seq=0
3449…	471.726548	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 53131 → 80 [SYN] Seq=0
3449…	471.727850	46.25.139.203	192.168.10.50	TCP	60	11501 → 80 [SYN] Seq=0 Win=2417 Len=0
3449…	471.729206	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 52425 → 80 [SYN] Seq=0
3449…	471.730491	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 13841 → 80 [SYN] Seq=0
3449…	471.731831	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 44653 → 80 [SYN] Seq=0
3449…	471.733148	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 26301 → 80 [SYN] Seq=0
3449…	471.734872	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 27257 → 80 [SYN] Seq=0
3449…	471.736072	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 56838 → 80 [SYN] Seq=0
3449…	471.737375	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 24895 → 80 [SYN] Seq=0
3449…	471.738653	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 13318 → 80 [SYN] Seq=0
3449…	471.740134	46.25.139.203	192.168.10.50	TCP	60	[TCP Port numbers reused] 40010 → 80 [SYN] Seq=0

Fonte: elaborado pelo autor

Neste cenário, as conexões da vítima se concentraram ao ataque SYN *Flood*. Assim, é possível observar a capacidade do atacante de realizar a negação de serviço.

A Figura 11 apresenta estatísticas da placa de rede da vítima neste cenário em específico. Através do utilitário iptraf, foi possível constatar a grande quantidade de pacotes TCP em comparação com outros tipos de pacotes, apesar de não estar sendo utilizada no momento.

Figura 11 – Estatísticas da placa de rede

```
iptraf-ng 1.1.4
Statistics for enp0s3

                  Total       Total    Incoming    Incoming    Outgoing    Outgoing
                Packets       Bytes     Packets       Bytes     Packets       Bytes
Total:           324338    14937168      323619    14886474         719       50694
IPv4:            324338    12995454      323619    12944760         719       50694
IPv6:                 0           0           0           0           0           0
TCP:             323619    12944760      323619    12944760           0           0
UDP:                270       20162           0           0         270       20162
ICMP:               449       30532           0           0         449       30532
Other IP:             0           0           0           0           0           0
Non-IP:               0           0           0           0           0           0

Total rates:        237.75 kbps             Broadcast packets:                    0
                      645 pps               Broadcast bytes:                      0

Incoming rates:     237.21 kbps
                      644 pps
                                          IP checksum errors:                   0
Outgoing rates:       0.54 kbps
                        1 pps
Elapsed time:   0:09
```

Fonte: elaborado pelo autor

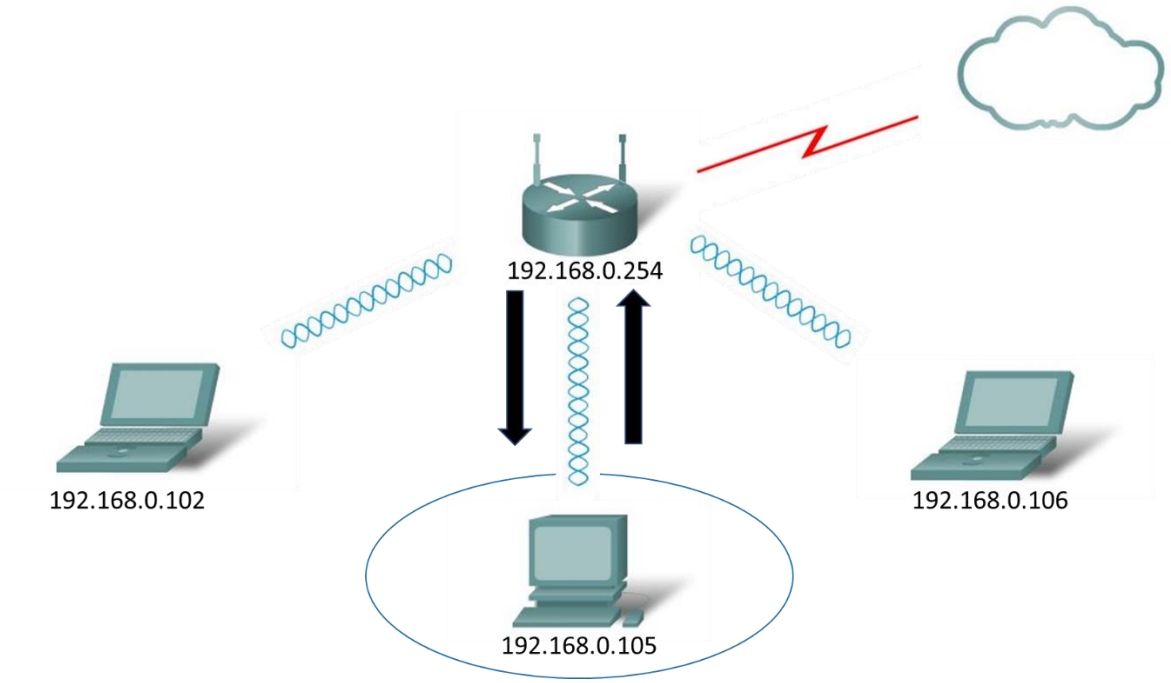
Este ataque durou aproximadamente 9 (nove) minutos. Caso outras sessões fossem abertas na máquina atacante, o ataque aumentaria proporcionalmente até que o limite físico dos componentes fosse atingido. Utilizando esse mesmo *framework*, seria possível realizar um Ataque de Negação de Serviço Distribuído (DDoS). Para obter dados reais para a pesquisa, foi necessário a preparação de um novo ambiente.

4.2 Preparação do ambiente

Para realizar a extração dos dados, se fez necessário montar dois cenários e realizar a negação de serviço através do SYN *Flood* em um deles.

O primeiro ambiente, cujo os dados de funcionamento comum da rede foram coletados, é de perfil doméstico e satisfatório para alimentar o algoritmo. A Figura 12 mostra como estava esse ambiente.

Figura 12 – Cenário do comportamento normal



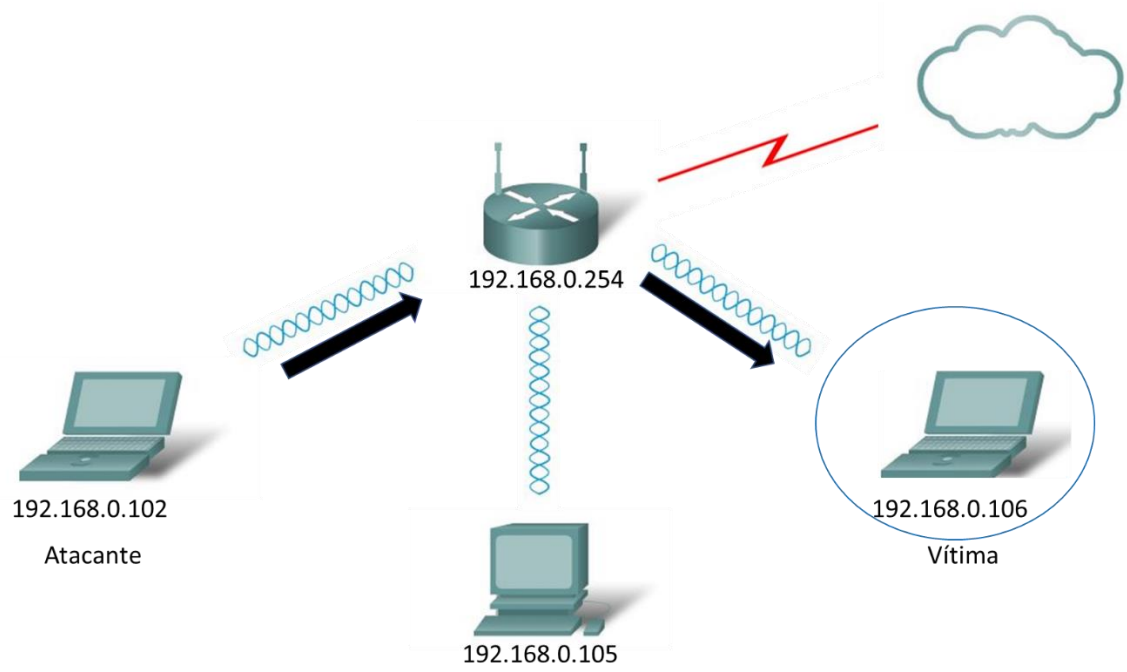
Fonte: elaborado pelo autor

Na rede, haviam diferentes dispositivos com acesso à internet. Todos utilizavam o mesmo roteador como *gateway*. O dispositivo monitorado era um *desktop* com Windows 10 64 bits, circulado na Figura.

O segundo cenário, de ataque, é similar ao anterior. Enquanto a mesma rede doméstica trafegava normalmente, uma máquina virtual Kali Linux foi utilizada como atacante e a vítima era um *notebook*.

Neste ambiente, a vítima fazia acessos normais enquanto sofria o ataque de negação de serviço. Foi nela, um *notebook* com Windows 10 64 bits, que o tráfego foi capturado. Está identificada pelo círculo na Figura 13.

Figura 13 – Cenário do ataque



Fonte: elaborado pelo autor

No cenário normal, mais de 100 mil pacotes foram capturados em aproximadamente 5 (cinco) minutos. Já no cenário de ataque, 1 milhão de pacotes foram capturados em torno de 16 (dezesseis) minutos.

Os dados, após capturados, são tratados e, na sequência, injetados no kNN para que o algoritmo realize a classificação como normal ou ataque. Esse processo está detalhado ao longo do Capítulo 5.

5 DETALHES DE IMPLEMENTAÇÃO

O experimento é dividido nas seguintes etapas, conforme fundamentado na Seção 2.3.

a) Coleta dos dados;
b) Tratamento dos dados;
c) Implementação do código (análise); e
d) Execução e testes.

5.1 Coleta dos dados

Para alimentar e testar o algoritmo, é necessário bases de dados onde possam ser reconhecidos os padrões de comportamento normal e durante o ataque.

Para a captura dos pacotes de comportamento normal, um dos dispositivos na rede foi selecionado para captura de pacotes. Foi utilizado o *software* Wireshark durante a utilização normal do *host* e, do mesmo modo, durante o ataque, gerando arquivos de extensão pcap contendo a captura.

A Figura 14 mostra alguns pacotes capturados durante o comportamento normal, convertidos para CSV utilizando o próprio *software* Wireshark.

Figura 14 – Dados coletados do comportamento normal

```
No.,"Time","Source","Destination","Protocol","Length","Info"
1,"0.000000","192.168.0.105","200.147.99.138","TCP","55","51739 > 443 [ACK] Seq=1 Ack=1 Win=258 Len=1 [TCP segment of a reassembled PDU]"
2,"0.033736","200.147.99.138","192.168.0.105","TCP","66","443 > 51739 [ACK] Seq=1 Ack=2 Win=262 Len=0 SLE=1 SRE=2"
3,"0.274807","192.168.0.105","172.217.28.226","TCP","55","51741 > 443 [ACK] Seq=1 Ack=1 Win=258 Len=1 [TCP segment of a reassembled PDU]"
4,"0.287424","172.217.28.226","192.168.0.105","TCP","66","443 > 51741 [ACK] Seq=1 Ack=2 Win=247 Len=0 SLE=1 SRE=2"
5,"1.169117","192.168.0.105","188.172.219.136","TCP","78","49676 > 5938 [PSH, ACK] Seq=1 Ack=1 Win=255 Len=24"
6,"1.497985","188.172.219.136","192.168.0.105","TCP","60","5938 > 49676 [ACK] Seq=1 Ack=25 Win=1026 Len=0"
7,"1.574007","192.168.0.105","8.43.72.62","TCP","55","51671 > 443 [ACK] Seq=1 Ack=1 Win=64800 Len=1 [TCP segment of a reassembled PDU]"
8,"1.578834","192.168.0.105","200.147.166.107","TCP","55","51742 > 443 [ACK] Seq=1 Ack=1 Win=258 Len=1 [TCP segment of a reassembled PDU]"
9,"1.578862","192.168.0.105","23.43.115.95","TCP","55","51570 > 443 [ACK] Seq=1 Ack=1 Win=257 Len=1 [TCP segment of a reassembled PDU]"
10,"1.592956","200.147.166.107","192.168.0.105","TCP","66","443 > 51742 [ACK] Seq=1 Ack=2 Win=140 Len=0 SLE=1 SRE=2"
11,"1.734201","8.43.72.62","192.168.0.105","TCP","60","443 > 51671 [ACK] Seq=1 Ack=2 Win=44160 Len=0"
12,"1.734855","192.168.0.105","54.230.57.84","TCP","55","51725 > 443 [ACK] Seq=1 Ack=1 Win=255 Len=1 [TCP segment of a reassembled PDU]"
13,"1.741043","54.230.57.84","192.168.0.105","TCP","66","443 > 51725 [ACK] Seq=1 Ack=2 Win=143 Len=0 SLE=1 SRE=2"
14,"1.777096","192.168.0.105","23.43.115.95","TCP","55","51627 > 443 [ACK] Seq=1 Ack=1 Win=253 Len=1 [TCP segment of a reassembled PDU]"
15,"1.815551","23.43.115.95","192.168.0.105","TCP","66","443 > 51570 [ACK] Seq=1 Ack=2 Win=424 Len=0 SLE=1 SRE=2"
16,"1.906938","192.168.0.105","172.217.28.234","TCP","55","51747 > 443 [ACK] Seq=1 Ack=1 Win=255 Len=1 [TCP segment of a reassembled PDU]"
17,"1.932323","172.217.28.234","192.168.0.105","TCP","66","443 > 51747 [ACK] Seq=1 Ack=2 Win=248 Len=0 SLE=1 SRE=2"
18,"1.990777","192.168.0.105","23.43.115.95","TCP","55","51571 > 443 [ACK] Seq=1 Ack=1 Win=256 Len=1 [TCP segment of a reassembled PDU]"
19,"2.021094","23.43.115.95","192.168.0.105","TCP","66","443 > 51627 [ACK] Seq=1 Ack=2 Win=283 Len=0 SLE=1 SRE=2"
20,"2.118439","192.168.0.105","8.43.72.54","TCP","55","51750 > 443 [ACK] Seq=1 Ack=1 Win=64240 Len=1 [TCP segment of a reassembled PDU]"
```

Fonte: elaborado pelo autor

Esses pacotes trazem consigo todas as informações, tais como protocolo, IP de origem, IP de destino, porta, tamanho, conteúdo, entre outros.

Alguns experimentos foram realizados a partir desses dados, conforme é apresentado na Tabela 5. Não obstante, esse método se aproxima à análise de assinatura, onde especificações dos pacotes são tratados. A intenção do trabalho é apresentar uma solução que possa detectar anomalias na rede.

Tabela 5 - Testes iniciais da análise dos pacotes

EXP	Qtdd A	Qtdd N	K	Teste	COL1	COL2	COL3	COL4	COL5	COL6	COL7	ERRO	ACERTO	Obs
0	65176	62062	3	50%	Length	Destination Port	Flag	Seq	Type	-	-	0,000299	99,970100%	SYNs Somados
1	65176	62062	3	50%	Length	Destination Port	Flag	Seq	Type	-	-	0,000000	100,000000%	
2	592	408	3	50%	Length	Source Port	Seq	Win	Type	-	-	0,816000	18,400000%	Apenas SYN
3	592	408	3	50%	Length	Source Port	Destination Port	Type	-	-	-	0,816000	18,400000%	Apenas SYN
4	65176	62062	3	50%	Length	Source Port	Destination Port	Cod Flag	Seq	Win	Type	0,000990	99,901000%	
5	65176	62062	3	50%	Length	Source Port	Destination Port	Cod Flag	Seq	Win	Type	0,000990	99,901000%	
6	65176	62062	3	50%	Length	Source Port	Destination Port	Cod Flag	Seq	Win	Type	0,000990	99,901000%	
7	65176	62062	4	50%	Length	Source Port	Destination Port	Cod Flag	Seq	Win	Type	0,000990	99,901000%	
8	65176	62062	15	50%	Length	Source Port	Destination Port	Cod Flag	Seq	Win	Type	0,001006	99,899400%	
9	65176	62062	25	50%	Length	Source Port	Destination Port	Cod Flag	Seq	Win	Type	0,001006	99,899400%	
10	65176	62062	3	20%	Length	Source Port	Destination Port	Cod Flag	Seq	Win	Type	0,000079	99,992100%	

Fonte: Elaborado pelo autor

Para então os objetivos, foi necessário extrair informações dessa base e tratá-las, conforme explica a seção 5.2.

5.2 Tratamento dos dados

Para que o algoritmo utilizado funcione, é necessário que os dados capturados sejam filtrados, padronizados e convertidos em numéricos. Mais do que isso, a correta extração e tratamento dos dados é essencial para que o Machine Learning possa realizar devidamente sua função.

Utilizou-se o código disponível no Apêndice B para gerar as informações abaixo:

- Quantidade total de pacotes;
- Quantidade de pacotes TCP;
- Quantidade de pacotes UDP;

- Proporção de pacotes TCP/UDP;
- Tamanho médio do pacote TCP;
- Quantidade de fluxos TCP;
- Tempo média de chegada de pacotes;
- *Flag* TCP SYN;
- *Flag* TCP SYN+ACK;
- *Flag* TCP ACK; e
- *Flag TCP PSH.*

A Figura 15 exibe algumas linhas de uma das bases de dados utilizada durante os experimentos. Ela é o resultado do tratamento e extração de informações dos arquivos pcap gerados anteriormente.

Figura 15 – Exemplo de base de dados

1359	1350	1	1350.00000000000000000000	.05674281503316138540	675	.00073464827078734363	675	0	2	0	1			
1518	1503	4	375.75000000000000000000	2.12343234323432343234	753	.00065706719367588932	753	0	5	0	1			
1536	1533	1	1533.00000000000000000000	6.21824104234527687296	752	.00065022005208333333	752	2	16	0	1			
1503	1503	1	1503.00000000000000000000	0	752	.00066509381237524950	752	0	0	0	1			
1510	1509	1	1509.00000000000000000000	0	755	.00066008675496688741	755	0	0	0	1			
1409	1400	1	1400.00000000000000000000	.52949538024164889836	700	.00070805110007097232	700	0	2	0	1			
1491	1486	1	1486.00000000000000000000	.00067204301075268817	743	.00066994366197183098	743	0	2	0	1			
1505	1497	1	1497.00000000000000000000	.62300531914893617021	749	.00066429102990033222	749	0	4	0	1			
1530	1508	21	71.80952380952380952380	.00066181336863004632	754	.00065306143790849673	754	0	2	0	1			
1491	1490	1	1490.00000000000000000000	0	745	.00066966264252179745	745	0	0	0	1			
1493	1490	1	1490.00000000000000000000	0	745	.00066883188211654387	745	0	0	0	1			
1399	1397	1	1397.00000000000000000000	.00071479628305932809	699	.00071401143674052894	699	0	2	0	1			
1592	1478	1	1478.00000000000000000000	59.99748585795097423004	739	.00062768027638190954	739	0	80	0	1			
1491	1488	2	744.00000000000000000000	0	744	.00066974513749161636	744	0	0	0	1			
1504	1496	2	748.00000000000000000000	.23200000000000000000	748	.00066300066489361702	748	0	2	0	1			
1983	1482	497	2.98189134808853118712	.02021563342318059299	741	.00050282198688855269	741	0	1	0	1			
1479	1474	1	1474.00000000000000000000	.02098849018280297901	737	.00067518593644354293	737	0	1	0	1			
1608	1472	1	1472.00000000000000000000	82.27753578095830740510	736	.00061916791044776119	736	0	125	0	1			
32	1	16	.06250000000000000000	.56250000000000000000	0	.02818100000000000000	0	0	16	0	6			
1414	1410	1	1410.00000000000000000000	.00070821529745042492	705	.00070356859971711456	705	0	2	0	1			

Fonte: elaborado pelo autor

Dentro de *Machine Learning*, tão importante quanto a escolha do algoritmo é o trabalho a ser feito com os dados. As primeiras extrações aplicadas no trabalho apenas formatavam os pacotes para serem interpretados no kNN. Todavia, essa técnica não seria suficiente para analisar o ataque de negação de serviço.

Para ser eficaz na detecção, através do EDA (*Exploratory Data Analysis*) como cita Shutt e O'Neil (2014, p. 36-37), os arquivos de monitoramento da rede, tanto normal quanto ataque, foram explorados e gerou-se estatísticas da rede a partir dos mesmos.

Foram gerados, além da base inicial de pacotes, três outras bases, variando as colunas e o tempo do tráfego. Utilizou-se o código disponível no Apêndice C (para 2 segundos, e adaptado para os demais) para fracionar os arquivos, que foram analisados para gerar as entradas na base do kNN. Foram as frações de tempo:

a) 10 segundos;
b) 4 segundos;
c) 2 segundos.

Durante os experimentos, foram testadas diferentes colunas, em diferentes quantidades e disposição gerais das mesmas. Isso influenciou diretamente os resultados. Na seção 5.3, é explicado como que as colunas são interpretadas e como adaptar o código para recebe-las.

5.3 Implementação do Machine Learning

Nesta seção o código é brevemente explicado, com ênfase nos pontos necessários para a interpretação da base nos diferentes experimentos.

Foi empregado o código sugerido pelo Harrington (2012, p. 18-32) para Python, onde o kNN é o algoritmo de *Machine Learning* adotado. O mesmo foi implementado em Python 2.7.15. Além de seguir suas recomendações, a escolha se baseia sobre o cenário atual. A linguagem de programação Python está se estabelecendo como uma das linguagens mais populares para computação. Segundo Pedregosa (2011, p. 2), a linguagem é cada vez utilizada, e não apenas em ambientes acadêmicos, mas também na indústria.

Um dos elementos fundamentais do programa é a forma com que lida com os dados, antes mesmo de interpretá-los. Para isso, foi empregado o Numpy. Confome Van der Walt (2011, p. 1), Numpy é a estrutura de dados de base usada para dados e parâmetros do modelo. Os dados de entrada são apresentados como matrizes numpy (de *n* dimensões), integrando-se perfeitamente a outras bibliotecas científicas do Python. Ele torna mais fácil (e ágil) a manipulação dos dados, tornando o programa com mais desempenho e menor consumo de memória.

O código está disponível nos apêndices do trabalho. O trecho apresentado abaixo possui todo o mecanismo central do kNN que é realizado quando acionado a função classify0, além das declarações, importação de módulos e definições para funcionamento do programa.

```
from numpy import *
import operator
from os import listdir
import kNN

def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] =
classCount.get(voteIlabel,0) + 1
    sortedClassCount = sorted(classCount.iteritems(),
key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]
```

A função file2matrix, apresentada na sequência, é responsável por ler a base de dados e adaptá-lo para que possa ser utilizado pelo *Machine Learning*. O arquivo é carregado e as informação são levadas ao *array* (estrutura de vetores e matrizes). O número de colunas que será interpretada do arquivo está definida nesse trecho.

```
def file2matrix(filename):
    dictionary={'Ataque':1, 'Normal':0}
    fr = open(filename)
    arrayOLines = fr.readlines()
    numberOfLines = len(arrayOLines)
    returnMat = zeros((numberOfLines,7))
    classLabelVector = []
    index = 0
    for line in arrayOLines:
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index,:] = listFromLine[0:7]
        if(listFromLine[-1].isdigit()):
            classLabelVector.append(int(listFromLine[-1]))
        else:

classLabelVector.append(dictionary.get(listFromLine[-1]))
        index += 1
    return returnMat,classLabelVector
```

A função seguinte, autoNorm, é responsável pela normalização dos dados. Quando se analisa estatisticamente um conjunto de dados, valores se diferem da maioria dos dados por serem muito elevados ou baixos podendo atrapalhar a análise como um todo. A normalização garante que esses pontos não irão prejudicar a análise desse conjunto.

```
def autoNorm(dataSet):
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
    normDataSet = zeros(shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - tile(minVals, (m,1))
    normDataSet = normDataSet/tile(ranges, (m,1))
    return normDataSet, ranges, minVals
```

A última função, ThreatTest, é a responsável pela execução do código. Ela convoca as funções citadas anteriormente, realiza a interpretação e testes, exibindo o resultado ao fim.

```
def ThreatTest():
    hoRatio = 0.20
    ThreatDataMat,ThreatLabels = file2matrix('kNN.input')
    normMat, ranges, minVals = autoNorm(ThreatDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m*hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify0(normMat[i,:],normMat[numTestVecs:m,:],
ThreatLabels[numTestVecs:m],3)
        print "the classifier came back with: %d, the real answer is: %d" %
(classifierResult, ThreatLabels[i])
        if (classifierResult != ThreatLabels[i]): errorCount += 1.0
    print "the total error with rate is: %f" %
(errorCount/float(numTestVecs))
    print errorCount
```

Quando o código é carregado, ao executar a função ThreatTest, o algoritmo começa a testar e exibe no terminal do Python os resultados obtidos, conforme explicado na Seção 5.4.

5.4 Execução e testes

Após todas as etapas de preparação citadas previamente, todos os requisitos para testar o algoritmo estão concluídos. Ao longo do desenvolvimento, foram feitos diferentes testes com diferentes entradas da base de dados e diferentes definições do código.

De maneira geral, mudou-se a quantidade de linhas de entrada, o valor do número de vizinhos levado em consideração pelo kNN e as colunas obtidas do monitoramento do tráfego de rede.

Esse modelo segue a linha de identificação por assinatura, onde os pacotes são analisados individualmente e então comparados ao modelo. Todavia, o objetivo é realizar a detecção a partir da análise de anomalias. Para tal, gerou-se estatísticas

do funcionamento da rede, como explicado na seção 5.2, e os testes foram realizados novamente.

Com os dados obtidos do fluxo da rede em porções de 2 segundos, tanto do cenário normal quanto do ataque, foram executados diversos testes. Eles estão resumidos na Tabela 6.

Tabela 6 – Testes de acerto

EXP	Qtdd. pacotes	Pact.TCP	Pact. UDP	Prop. TCP/UDP	Tamanho medio pact. TCP	Fluxos TCP	Tempo médio chegada	SYN	SYN+ ACK	ACK	PSH	Erro	Acerto
1	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	0,000000	**100,0000%**
2	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	NÃO	0,000000	**100,0000%**
3	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	NÃO	NÃO	0,000000	**100,0000%**
4	SIM	SIM	SIM	SIM	SIM	SIM	SIM	SIM	NÃO	NÃO	NÃO	0,000000	**100,0000%**
5	SIM	SIM	SIM	SIM	SIM	SIM	SIM	NÃO	NÃO	NÃO	NÃO	0,000000	**100,0000%**
6	NÃO	SIM	NÃO	SIM	SIM	SIM	SIM	NÃO	NÃO	NÃO	NÃO	0,009804	**99,0196%**
7	SIM	SIM	NÃO	SIM	SIM	NÃO	SIM	NÃO	NÃO	NÃO	NÃO	0,000000	**100,0000%**
8	NÃO	SIM	NÃO	SIM	SIM	NÃO	SIM	NÃO	NÃO	NÃO	NÃO	0,000000	**100,0000%**
9	NÃO	SIM	NÃO	SIM	SIM	NÃO	SIM	SIM	NÃO	SIM	NÃO	0,000000	**100,0000%**
10	SIM	SIM	SIM	NÃO	SIM	SIM	SIM	SIM	NÃO	SIM	NÃO	0,000000	**100,0000%**
11	SIM	SIM	SIM	NÃO	SIM	SIM	SIM	SIM	NÃO	SIM	SIM	0,000000	**100,0000%**
12	NÃO	NÃO	NÃO	SIM	SIM	SIM	SIM	NÃO	NÃO	NÃO	NÃO	0,044776	**95,5224%**
13	NÃO	SIM	NÃO	SIM	SIM	SIM	SIM	SIM	SIM	SIM	NÃO	0,000000	**100,0000%**
14	NÃO	SIM	NÃO	SIM	SIM	NÃO	SIM	NÃO	NÃO	NÃO	NÃO	0,007463	**99,2537%**
15	NÃO	NÃO	NÃO	SIM	SIM	NÃO	SIM	NÃO	NÃO	NÃO	NÃO	0,044776	**95,5224%**

Fonte: Elaborado pelo autor

Ao longo dos 15 experimentos, testou-se a eficácia na detecção a partir de diferentes combinações. Dos dados gerados, utilizou-se a quantidade total de pacotes; quantidade de pacotes TCP; quantidade de pacotes UDP; proporção de pacotes TCP/UDP; tamanho médio do pacote TCP; quantidade de fluxos TCP; Tempo média de chegada de pacotes; *Flag* TCP SYN; *Flag* TCP SYN+ACK; *Flag* TCP ACK; e *Flag* TCP PSH. Na sequência da tabela 6, há o valor do erro obtido e a porcentagem de acerto.

Ao final de cada execução do algoritmo, o terminal exibia o percentual e quantia de erros, o que permitiu a estimativa dos resultados. A Figura 16 exibe o resultado referente ao experimento 12 da Tabela 5.

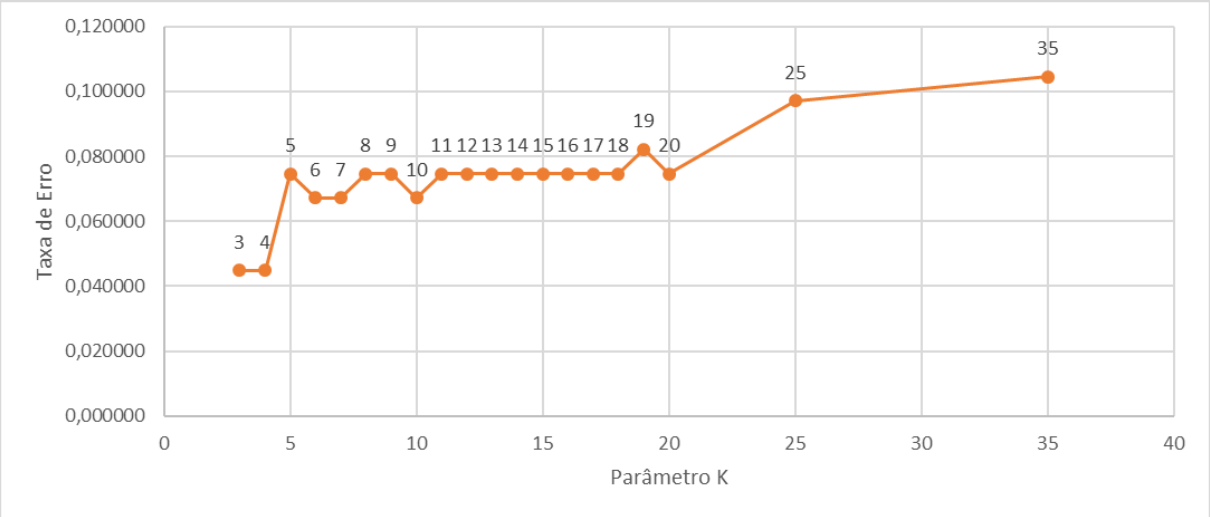
Figura 16 – Teste do classificador

```
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 6, the real answer is: 6
the classifier came back with: 1, the real answer is: 1
the classifier came back with: 6, the real answer is: 6
the total error with rate is: 0.044776
6.0
>>>
```

Fonte: elaborado pelo autor

O valor de *k* utilizado ao longo de todo experimento citado na tabela 5 foi igual a 3, que determina a quantidade de vizinhos. Utilizando a base de 2 segundos e as colunas do experimento 15 (proporção TCP/UDP, tamanho médio do pacote TCP e Média de tempo chegada), testou-se o resultado para diferentes valores de k, conforme exibido no Gráfico 3. Os valores com maior acerto foram 3 e 4, com 0,044776 de erro, seguido dos valores 6,7 e 10, com 0,067164 de erro. Vale ressaltar que quanto maior o número de *k*, maior é o tempo que o algoritmo leva para fazer os testes. Para evitar eventuais empates de números de vizinhos, costuma-se empregar valores ímpares para *k.*

Gráfico 3 – Erro em função de K



Fonte: Elaborado pelo autor

Por fim, a quantidade de arquivos disponíveis e rotulados para a preparação do kNN também é significante. Utilizando uma base que foi gerada a partir do tráfego de rede na porção de 4 segundos, mediu-se a taxa de erros com diferentes valores percentuais de dados utilizados no teste.

Tabela 6 – Percentual de dados para teste

% Teste	Erro
50	0,294118
40	0,240741
30	0,150000
20	0,111110
15	0,050000
10	0,000000
5	0,000000

Fonte: Elaborado pelo autor

A Tabela 6 deixa evidente que quanto maior a porção da base disponível para teste, maior tende a ser o erro do algoritmo.

6 CONSIDERAÇÕES FINAIS

A segurança da informação é essencial para a proteção de usuários domésticos e de corporações, pois protegem estes de ataques e riscos. Um dos ataques, que atua causando indisponibilidade do sistema, é o ataque de Negação de Serviço.

Este pode ocorrer de diversas maneiras. A mais comum, onde sobrecarrega-se o sistema por meio do alto número de requisições, faz com que conexões legítimas sejam prejudicadas. Há mecanismos de segurança de informação que se baseiam em assinaturas para identificar ataques, porém são ineficazes contra novas técnicas. Os outros modelos, baseados em anomalia, são complexos de serem implementados e pouco adotados.

Para solucionar esse problema, estudou-se aplicar a tecnologia de *Machine Learning* a fim de, com alta precisão, fosse possível classificar corretamente as conexões durante um ataque SYN *Flood*, mas de modo flexível, tratando como uma anomalia.

Utilizando o algoritmo kNN, foi possível com alta precisão realizar essa distinção. Os piores resultados obtiveram 95,52% de acerto, enquanto os melhores ultrapassaram 99%.

Para aplicar essa resolução em um cenário real de uma organização, se faz necessário criar uma entrada dos dados do tráfego em tempo real, bem como possuir uma capacidade de processamento dimensionada para a infraestrutura em questão.

A base de dados do ataque possuía o comportamento de um único tipo de negação de serviço, o SYN *Flood*. A natureza desse ataque eleva o número de requisições TCP solicitando abertura de conexão (*flag* SYN). Esse comportamento, quando comparado ao de uma rede doméstica, como utilizada no experimento, torna menos complexo identificar com o *Machine Learning* quais são acessos legítimos e quais são não legítimos. Em um cenário onde o volume de requisições TCP é normalmente alto, a Análise dos Dados Exploratória (EDA) precisa levar em consideração outros pontos, como horários de picos, eventos programados, entre outros. Do mesmo modo, em caso de outros tipos de ataque dessa mesma natureza,

a base de dados inicial precisa conter diferentes amostras para poder classificar e detectar corretamente.

Há outros algoritmos de *Machine Learning*, que podem ser empregados e obter resultados equivalentes ou até superiores, porém com menor requisito de processamento e melhor aproveitamento de memória, como cita Doshi (2018, p. 6) e Harrington (2012, p. 19).

Neste trabalho, empregou-se a linguagem de programação Python. Para obter melhor desempenho de processamento e menor tempo de análise, pode-se investigar o uso do Cyton, combinação da linguagem C com Python, que permite o processamento das informações de maneira mais eficaz, e que poderia ser utilizado. Van der Walt (2011, p. 8) afirma que sua utilização explora de maneira mais eficiente os efeitos de *cache*.

Como trabalhos futuros, sugere-se explorar o funcionamento de outros algoritmos de *Machine Learning*, inclusive os não-supervisionados. Além disso, obter dados de um cenário com grandes fluxos de informação, para assim se aproximar do comportamento diário de um equipamento em uso e poder explorar mais profundamente estas informações. Por fim, além da detecção, pode-se desenvolver uma aplicação que possa proteger os equipamentos caso detectado um ataque de negação de serviço, independente da técnica utilizada.

REFERÊNCIAS

APTHORPE, Noah; FEAMSTER, Nick; DOSHI, Rohan. **Machine learning DDoS detection for consumer Internet of Things devices.** 11 de abril de 2018. Disponível em: <https://arxiv.org/abs/1804.04159>. Acesso em: 7 out. 2018, às 10h28min.

CARISSIMI, Alexandre da Silva; GRANVILLE, Lisandro Zambenedetti; ROCHOL, Juergen. **Redes de computadores**. Porto Alegre: Editora Artmed S.A., 2008.

CERT. BR**. Cartilha de segurança para Internet**, versão 4.0. São Paulo: Comitê Gestor da Internet no Brasil, 2012.

CUNHA, Lilian M. L. **Análise de agrupamento**, Estatística aplicada III, 2017. Disponível em <https://edisciplinas.usp.br/pluginfile.php/3498333/mod_resource/content/0/AULA4-2017.pdf>. Acesso em 15 nov 18, às 16h34min.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de banco de dados**, 6ª edição. São Paulo: Editora PEARSON, 2011.

FACHIN, Odília. **Fundamentos da metodologia**, 5ª edição. São Paulo: Editora Saraiva, 2006.

FONTES, Edison. **Praticando a Segurança da Informação**, 6ª Edição. Rio de Janeiro: Editora Brasport, 2010.

FOROUZAN, Behrouz A. **Comunicação de dados e redes de computadores**, 4ª Edição. Porto Alegre: Editora Mc Graw Hil, 2010.

FOROUZAN, Behrouz A. **Protocolo TCP/IP**, 3ª Edição. Porto Alegre: Editora Mc Graw Hil, 2008.

HARRINGTON, Peter. **Machine Learning in Action**, 1ª Edição. Nova Iorque: Editora Manning, 2012.

KUROSE, James F; ROSS, Keith W. **Redes de computadores e a internet**, 5ª Edição. São Paulo: Editora Pearson, 2010.

LARSON, Eric; STEPHENS, Brian. **Web Servers, Security & Maintence**, 1ª Edição. Nova Jersey: Editora PH PTR, 2000.

LEARN, Scikit. **Nearest Neighbors.** 2018. Disponível em <https://scikit-learn.org/stable/modules/neighbors.html>. Acesso em 15 nov 18.

LYRA, Maurício Rocha. **Segurança e Auditoria em Sistemas de Informação**, 1ª Edição. Rio de Janeiro: Editora Ciência Moderna, 2008.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Técnicas de Pesquisa**, 7ª edição. São Paulo: Editora Atlas, 2009.

OPPEL, Andrew J. **Banco de Dados**, 1ª Edição. Rio de Janeiro: Editora Alta Books, 2009.

PEDREGOSA, Fabian; *et al*. **Scikit-learn:** machine learning in Python. 2 de janeiro 2011. Disponível em < https://dl.acm.org/citation.cfm?id=2078195>. Acesso em: 10 out. 2018, às 8h24min.

VAN DER WALT, S; COLBERT, S.C; VAROQUAUX, G. **The NumPy array:** a structure for efficient numerical computation. 2011. Disponível em: <https://hal.inria.fr/inria-00564007/document>. Acesso em: 12 out. 2018, às 18h24min.

SAS. **Machine Learning:** o que é e qual a sua importância? 2018. Disponível em <https://www.sas.com/pt_br/insights/analytics/machine-learning.html>. Acesso em 15 maio 18.

SCHUTT, Rachel; O'Neil, Cathy. **Doing Data Science**, 1ª Edição. Sebastopol: Editora O'Reilly, 2013.

STALLINGS, William. **Criptografia e segurança de redes**, 4ª Edição. São Paulo: Editora Pearson, 2008.

TANENBAUM, Andrew S. **Redes de Computadores**, 4ª Edição. Rio de Janeiro: Editora Campus, 2003.

APÊNDICES

Apêndice A – Programa do *Machine Learning* com kNN adaptado de Harrington

```
from numpy import *
import operator
from os import listdir
import kNN

def classify0(inX, dataSet, labels, k):
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX, (dataSetSize,1)) - dataSet
    sqDiffMat = diffMat**2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances**0.5
    sortedDistIndicies = distances.argsort()
    classCount={}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel,0)
+ 1
    sortedClassCount = sorted(classCount.iteritems(),
key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]

def file2matrix(filename):
    dictionary={'Ataque':1, 'Normal':6}
    fr = open(filename)
    arrayOLines = fr.readlines()
    numberOfLines = len(arrayOLines)
    returnMat = zeros((numberOfLines,4))
    classLabelVector = []
    index = 0
    for line in arrayOLines:
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index,:] = listFromLine[0:4]
        if(listFromLine[-1].isdigit()):
            classLabelVector.append(int(listFromLine[-1]))
        else:

classLabelVector.append(dictionary.get(listFromLine[-1]))
        index += 1
    return returnMat,classLabelVector

def autoNorm(dataSet):
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
```

```
    normDataSet = zeros(shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - tile(minVals, (m,1))
    normDataSet = normDataSet/tile(ranges, (m,1))
    return normDataSet, ranges, minVals

def ThreatTest():
    hoRatio = 0.20
    ThreatDataMat,ThreatLabels = file2matrix('base_2s.txt')
    normMat, ranges, minVals = autoNorm(ThreatDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m*hoRatio)
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult =
classify0(normMat[i,:],normMat[numTestVecs:m,:],ThreatLabels[n
umTestVecs:m],3)
        print "the classifier came back with: %d, the real
answer is: %d" % (classifierResult, ThreatLabels[i])
        if (classifierResult != ThreatLabels[i]): errorCount
+= 1.0
    print "the total error with rate is: %f" %
(errorCount/float(numTestVecs))
    print errorCount
```

Apêndice B – Shell Script para extração de dados

```
#!/bin/bash

### extract features
i=0

touch "base_$$.txt"

entradas=(${entradas[@]} `ls bases2/ | grep -v ".out"`)
for i in `echo ${!entradas[*]}`;do

     #FILENAME="bases2/minute-21-ataque.pcap"
     FILENAME="bases2/"${entradas[$i]}

     flag_ataq_nor=$(echo ${entradas[$i]} | grep ataque | wc -
l)
     if [ $flag_ataq_nor == 1 ]; then
          flag=1
     else
          flag=6
     fi

     echo " "
     echo "-------------------------"
     echo $FILENAME" flag "$flag

     interval=($(/usr/sbin/tcpdump ip -nr ${FILENAME} | cut -d'
' -f1 | cut -d: -f3))
     tcpflows=$(tstat ${FILENAME} | grep 'total TCP flows
analized :' | cut -d: -f2 | sed 's/ //g')
     udpflows=$(tstat ${FILENAME} | grep 'total UDP flows
analized :' | cut -d: -f2 | sed 's/ //g')
     totalvalidpackets=$(tstat ${FILENAME} | grep 'packets
seen' | cut -d' ' -f1)
     tcppackets=$(tstat ${FILENAME} | grep 'packets seen' | cut
-d' ' -f4)
     udppackets=$(tstat ${FILENAME} | grep 'packets seen' | cut
-d' ' -f8)
     if [ $udppackets == 0 ] && [ $tcppackets != 0 ]; then
          udppackets=1
     fi
     if [ $udppackets == 0 ] && [ $tcppackets == 0 ]; then
          udppackets=1
          tcppackets=1
     fi
     if [ $tcppackets == 0 ] && [ $udppackets != 0 ]; then
          tcppackets=1
     fi
     proportion=$(echo "$tcppackets / $udppackets" | bc -l)
```

```
     arraysize=${#interval[*]}

          tcp_packet_size_array=($(/usr/sbin/tcpdump ip and tcp
and port ! 53 -nr ${FILENAME} | grep length | awk -F length
'{print $2}' | sed 's/[a-zA-Z: /.]//g'))
     tcp_array_size=${#tcp_packet_size_array[*]}
     sum_tcp_packet_size="0";
     for size in ${tcp_packet_size_array[@]}; do
          sum_tcp_packet_size=$(echo "$size +
$sum_tcp_packet_size" | bc -l)
          #let i=i+1
          #echo $i" "$sum_tcp_packet_size

     done
     tcp_mean_packet_size=$(echo "$sum_tcp_packet_size /
$tcp_array_size" | bc -l)

     sum=0;
     for index in ${!interval[*]}; do
          tmpindex=$(echo "$index + 1" | bc)
          if [ "$tmpindex" -lt "$arraysize" ]; then
               result=$(echo "${interval[$tmpindex]} -
${interval[$index]}" | bc)
               sum=$(echo "$sum + $result" | bc -l)
          fi
     done

     #conta flags
     #SYN
     qtdde_SYN=$(tcpdump -r ${FILENAME} -n tcp[13] == 2 | wc -
l)
     #SYN+ACK
     qtdde_SYN_ACK=$(tcpdump -r ${FILENAME} -n tcp[13] == 18 |
wc -l)
     #ACK
     qtdde_ACK=$(tcpdump -r ${FILENAME} -n tcp[13] == 16 | wc -
l)
     #PSH
     qtdde_PSH=$(tcpdump -r ${FILENAME} -n tcp[13] == 8 | wc -
l)

     media=$(echo "$sum / $arraysize"  | bc -l)

     echo "$arraysize    $tcppackets    $udppackets
     $proportion    $tcp_mean_packet_size    $tcpflows $media
     $qtdde_SYN     $qtdde_SYN_ACK $qtdde_ACK     $qtdde_PSH
     $flag" >> base_$$.txt
     echo " "
done
```

Apêndice C – Shell Script para divisão dos arquivos pcap em intervalo de 2 segundos

```
#!/bin/bash

rm -rf bases2/
mkdir bases2

FILENAME_NORMAL=normal.pcap
FILENAME_ATAQUE=ataque.pcap

minuto_normal=50
minuto_ataque=50

duracao_normal=6
duracao_ataque=10

segundo_inicial=0
segundo_final=1

a=0
while [ $a -le $duracao_normal ]; do
     b=0

     while [ $b -le 29 ]; do
        editcap -F libpcap -A "2018-10-25
21:$minuto_normal:"$segundo_inicial -B "2018-10-25
21:$minuto_normal:"$segundo_final $FILENAME_NORMAL
bases2/"minute-"$a"_"$b"_"$segundo_final"-normal.pcap"
          let segundo_inicial=segundo_inicial+2
          let segundo_final=segundo_final+2
          let b=b+1
     done

     let minuto_normal=minuto_normal+1
     segundo_inicial=0
     segundo_final=1
     let a=a+1

done

segundo_inicial=0
segundo_final=1

a=0
while [ $a -le $duracao_ataque ]; do
     b=0
     segundo_ataque=0
     segundo_ataque_final=9
     while [ $b -le 29 ]; do
```

```
        editcap -F libpcap -A "2018-11-06
22:$minuto_ataque:"$segundo_inicial -B "2018-11-06
22:$minuto_ataque:"$segundo_final $FILENAME_ATAQUE
bases2/"minute-"$a"_"$b"_"$segundo_final"-ataque.pcap"
          let segundo_inicial=segundo_inicial+2
          let segundo_final=segundo_final+2
          let b=b+1
     done

     let minuto_ataque=minuto_ataque+1
     segundo_inicial=0
     segundo_final=1
     let a=a+1

done

ls -ltr bases2/

verifica_arqs_vazios=$(du bases2/* | grep '^4\s' | sed
's/4\s//g' | wc -l)
if [ $verifica_arqs_vazios != 0 ]; then
     echo "apagando por ser vazio "$(du bases2/* | grep '^4\s'
| sed 's/4\s//g')
     echo "enter para confirmar"
     read x
     rm -f $(du bases2/* | grep '^4\s' | sed 's/4\s//g')
fi

echo "fim"
```