INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

CAMPUS CATANDUVA

JOÃO QUERINO DOS REIS NETO E VICTOR MARCELO FERRAZ VEIGA ARAUJO

DESENVOLVIMENTO E APLICAÇÃO DE SCRIPTS PYTHON PARA TESTES DE SEGURANÇA EM REDES COMPUTACIONAIS

CATANDUVA - SP

2023

JOÃO QUERINO DOS REIS NETO E VICTOR MARCELO FERRAZ VEIGA ARAUJO

DESENVOLVIMENTO E APLICAÇÃO DE SCRIPTS PYTHON PARA TESTES DE SEGURANÇA EM REDES COMPUTACIONAIS

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Campus Catanduva, como requisito parcial para conclusão do curso Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Edivaldo Pastori Valentini

CATANDUVA - SP

2023

Dados Internacionais de Catalogação na Publicação (CIP)
Bibliotecária: Luiza Correia Lima Felix – CRB-8/10837

Reis Neto, João Querino dos.
R375d Desenvolvimento e aplicação de scripts Python para testes de segurança em redes computacionais / João Querino dos Reis Neto; Victor Marcelo Ferraz Veiga Araújo.
Catanduva, SP : IFSP, 2023.
50 f. : il.

Trabalho de Conclusão de Curso (Tecnólogo – Análise e Desenvolvimento de Sistema) – Instituto Federal de São Paulo, campus Catanduva, 2023.
Orientador: Prof. Me. Edivaldo Pastori Valentini.

1. Segurança da Informação. 2. Scripts Python. 3. Sniffer. 4. Detecção de ataques.
I. Araújo, Victor Marcelo Ferraz Veiga. II. Título.

CDD (23 ed.): 005.4

ATA N.º 12/2023 - TIF-CTD/DAE-CTD/DRG/CTD/IFSP

Ata de Defesa de Trabalho de Conclusão de Curso - Graduação

Na presente data realizou-se a sessão pública de defesa do Trabalho de Conclusão de Curso intitulado **DESENVOLVIMENTO E APLICAÇÃO DE SCRIPTS PYTHON PARA TESTES DE SEGURANÇA EM REDES COMPUTACIONAIS** apresentado pelos alunos **João Querino dos Reis Neto (AL3006336)** e **Victor Marcelo Ferraz Veiga Araujo (AL3006344)** do Curso **SUPERIOR EM ANÁLISE E DESENVOLIMENTO DE SISTEMAS,** Campus Catanduva. Os trabalhos foram iniciados às **13:30** pelo Professor presidente da banca examinadora, constituída pelos seguintes membros:

Membros	IES	Presença (Sim/Não)	Aprovação/Conceito (Quando Exigido)
Edivaldo Pastori Valentini (Presidente/Orientador)	IFSP-CTD	SIM	9.0
Fábio Luiz Viana (Examinador 1)	IFSP-CTD	SIM	9.0
Luis Fernando Castilho Maschi (Examinador 2)	IFSP-CTD	SIM	9.0

Observações:

A banca examinadora, tendo terminado a apresentação do conteúdo da monografia, passou à arguição dos candidatos. Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelos alunos, tendo sido atribuído o seguinte resultado:

[X] Aprovado(a) [ ] Reprovado(a) Nota (quando exigido): 9.0

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu lavrei a presente ata que assino juntamente com os demais membros da banca examinadora.

Canduva, 29 de novembro de 2023.

Avaliador externo: [ ]Sim [X]Não

Assinatura:

RESUMO

A segurança da informação é um tema crítico na sociedade atual, especialmente em ambientes corporativos, onde as redes podem estar vulneráveis a ataques cibernéticos. Este trabalho demonstrou um conjunto de *scripts* desenvolvidos pela linguagem de programação Python 3.12, para realização e avaliação de testes de segurança em redes computacionais. O objetivo foi criar ferramentas eficientes e acessíveis que permitam aos usuários entender a segurança de suas redes. O trabalho também buscou educar os usuários domésticos sobre a importância da segurança da informação e técnicas de segurança. Foram desenvolvidos *scripts* de segurança para criação e manipulação de pacotes interagindo nas camadas da arquitetura TCP/IP: enlace de dados, rede e transporte. Como resultados obtidos o trabalho desenvolveu e validou as seguintes ferramentas de segurança de redes: *Sniffer* para análise e captura do tráfego de dados da rede local, elaboração de cabeçalhos e *payloads* do protoco IP (Internet *Protocol*, ou Protocolo de Internet), varreduras e enumeração de portas abertas/fechadas (*port scanner*) na camada de transporte e uma ferramenta para análise e monitoramento da camada de rede para detecção de ataques de negação de serviço (DoS - *Denial of Service*).

Palavras-chave: Segurança da Informação, Scripts Python, Sniffer, LAN, Detecção de ataques.

ABSTRACT

Information security is a critical topic in today's society, especially in corporate environments, where networks may be vulnerable to cyberattacks. This work has demonstrated a set of *scripts* developed using the Python 3.12 programming language for conducting and evaluating security tests on computer networks. The aim was to create efficient and accessible tools that allow users to understand the security of their networks. The work also aimed to educate home users about the importance of information security and security techniques. Security *scripts* were developed for creating and manipulating packets that interact at the layers of the TCP/IP architecture: data link, network, and transport. As a result, the work developed and validated the following network security tools: a *Sniffer* for analyzing and capturing local network data traffic, crafting IP (Internet *Protocol*) headers and *payloads*, scanning and enumerating open/closed ports (*port scanner*) at the transport layer, and a tool for analyzing and monitoring the network layer to detect Denial of Service (DoS) attacks.

Keywords: Information Security, Python Scripts, Sniffer, LAN, Attack Detection.

LISTA DE ABREVIATURAS E SIGLAS

ARP	Address Resolution Protocol
DDoS	Distributed Denial-of-Service
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
NMAP	Network Mapper
RFC	Request for Comments
SYN	Synchronize do Three-way Handshake
TCP	Protocolo de Controle de Transmissão
UDP	User Datagram Protocol

SUMÁRIO

1 INTRODUÇÃO

Como descrito por (WHITMAN; MATTORD, 2011) as organizações passam por uma variedade de riscos cibernéticos, que vão de ataques de pessoas más-intencionadas, erros humanos e até falhas técnicas. Para se prevenir desses riscos é vital que a empresa tome medidas proativas. Para isso é implementado medidas de segurança, capacitação dos funcionários para identificar e responder a ameaças. Nesse contexto, a realização de testes de segurança em redes de computadores sistemas operacionais, é essencial para identificar vulnerabilidades e garantir a proteção adequada das informações.

A segurança em redes computacionais é uma preocupação em ascensão à medida que mais dispositivos são conectados à rede. É de extrema relevância que usuários tomem medidas com o objetivo de garantir que suas redes estejam seguras, usar senhas fortes e manter atualizado o *firmware* (Software que é armazenado em um dispositivo eletrônico e que controla o funcionamento deste dispositivo) do roteador são exemplos de medidas a serem tomadas para não negligenciar a segurança da rede doméstica (ZHANG et al., 2018). Neste trabalho de conclusão de curso, foi desenvolvido um conjunto de *scripts* de teste de segurança utilizando utilizando a linguagem Python, que foram analisados, testados e executados numa rede residencial.

Python é uma linguagem popular na área de segurança da informação, muito deve a sua simplicidade, facilidade de uso e versatilidade. Diversas ferramentas de segurança como *scanner* de vulnerabilidade, exploração de rede e programa de análise de tráfego foram desenvolvidas em Python, para isso são usadas várias bibliotecas como PyCrypto, Scapy e Socket que possibilitam usar criptografia, análise de protocolo e comunicação de rede (LIM, 2017).

Os scripts foram criados para avaliar a segurança da rede em termos de vulnerabilidades de rede. O trabalho se concentrou em utilizar técnicas populares de teste de penetração para avaliar a segurança da rede e exemplificar o quão vulneráveis as pessoas estão em uma rede *LAN* (Local Area Network). O trabalho foi concluído obtendo como resultados o desenvolvimento de três *scritps* por meio da linguagem Python, sendo estes: a) *sniffers* para análise de pacotes de redes, b) port scanner ou varreduras e descoberta de portas e aplicações abertas e c) uma aplicação com recursos de monitoramento do recebimento de pacotes de redes e na geração de alertar na detecção de anomalias num determinado *host* da rede. Espera-se que os *scripts* contribuam para avaliar a segurança de redes computacionais, bem como recomendações para melhorar a segurança dessas redes.

1.1 Objetivo geral

O objetivo deste trabalho é criar um conjunto de *scripts* que permitam aos usuários testar a segurança de sua rede doméstica de forma eficiente e eficaz, além de entender fundamentos de redes.

O objetivo da pesquisa é o desenvolvimento de *scripts* que possibilitem os usuários a detectar vulnerabilidades em dispositivos conectados em redes computacionais públicas de maneira simples e fácil, necessitando fazer apenas algumas passagens de parâmetros. Um *script* em Python irá realizar uma série de ações e varreduras buscando vulnerabilidades na conexão do cliente e servidor, e com isso alertar o usuário sobre o status de segurança dos *hosts* conectados.

1.1.1 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Desenvolver *scripts* independentes em Python, que sejam aplicados e executados numa rede computacional e seja possível verificar as portas de comunicação que estejam abertas e capturar pacotes que estejam trafegando de um *host* para outro;
- Exemplificar o quão fácil seria uma interceptação de pacotes de uma aplicação quando se tem acesso a uma rede privada ou não, dando um exemplo da importância da atenção ao conectar em uma rede sem senha;
- Aumentar a conscientização de segurança para desenvolvedores das aplicações que usam comunicação por rede, em especial com aplicações que operam dados sensíveis dos cliente dos usuários, como CPF, endereços, senhas, e-mail e qualquer outra informação que possa ser de interesse de terceiros mal intencionados;
- Evitar (ou ao menos reduzir) roubo de dados pessoais via Wi-Fi, visto que com tecnologia novas permitem ter conexões com a internet a partir de qualquer parte do planeta e é cada vez mais raro uma área urbana que não tenha uma rede Wi-Fi disponível.

1.2 Organização do Trabalho

Este trabalho está organizado de acordo com os seguintes capítulos:

• No Capítulo 2 é apresentada a fundamentação teórica.

• No Capítulo 3 são discutidos os trabalhos correlacionados com a linha deste estudo com ênfase nos testes de segurança computacional.

• No Capítulo 4 apresenta as etapas do desenvolvimento, descrição das bibliotecas da linguagem Python e os *scripts* desenvolvidos com experimentos, funcionalidades e execuções dos resultados obtidos.

• No Capítulo 5, além da conclusão, também são apresentadas as dificuldades encontradas e considerações finais.

1.3 Justificativa

A segurança da informação é um tema de grande importância na sociedade atual, visto que a tecnologia e a internet se tornaram parte integrante da vida cotidiana das pessoas. A segurança da informação é um desafio constante e a realização de testes de segurança é uma maneira essencial de garantir que as informações estejam protegidas contra ataques cibernéticos. Em ambientes domésticos, é comum que as redes não sejam configuradas adequadamente ou não tenham medidas de segurança suficientes, tornando-as vulneráveis a ataques cibernéticos. A maioria dos usuários domésticos não têm conhecimento suficiente sobre técnicas de segurança e, muitas vezes, não realizam testes de segurança em suas redes. (CERT.br; NIC.br, 2012)

Este trabalho se justifica pela necessidade de desenvolver ferramentas acessíveis e eficientes para a realização de testes de segurança em redes computacionais. Com o conjunto de *scripts* de teste de segurança em Python, os usuários poderão avaliar a segurança de sua rede doméstica de forma eficaz e tomar medidas para proteger suas informações pessoais e privacidade.

O trabalho também se justifica pela importância de educar os usuários domésticos sobre a importância da segurança da informação e técnicas de segurança. O conjunto de *scripts* desenvolvidos neste trabalho não apenas fornecerá uma avaliação da segurança da rede doméstica, mas também fornecerá recomendações para melhorar a segurança, ajudando os usuários a entender melhor a importância da segurança da informação em seus ambientes domésticos.

Por fim, a realização deste trabalho é justificada pela relevância do tema para a sociedade como um todo. A segurança da informação é uma questão crítica e a conscientização sobre as ameaças cibernéticas e técnicas de segurança é fundamental

para garantir a proteção de informações pessoais e empresariais.

No gráfico, ilustrado na Figura 1, são demonstrados os totais de ataques notificados ao CERT.br até em setembro do ano de 2023.

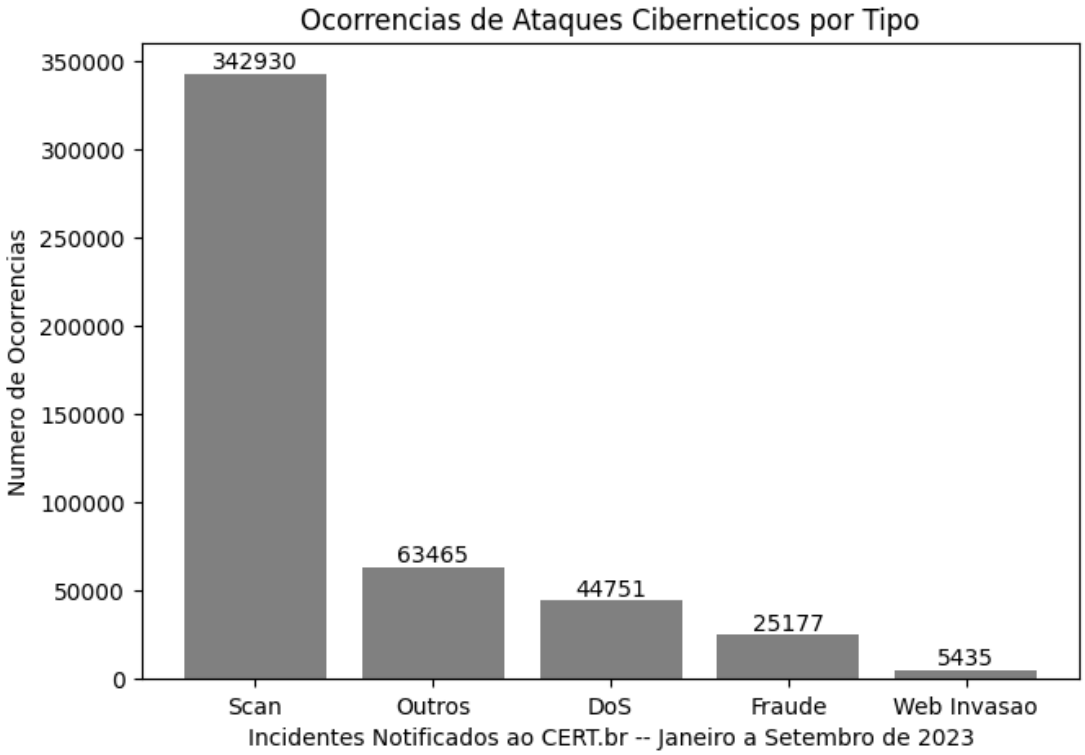


Figura 1 – Gráfico de tentativas de ataques

Fonte: Adaptado de: CERT.br (2023)

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Como funciona uma rede de computadores?

Neste trabalho será abordada a comunicação Cliente Servidor, mas há outras menos comuns. De acordo com Servidores e clientes são aparelhos com um sistema operacional vinculados à rede. Com esse vínculo buscando ser o mais transparente possível para permitir que um maior número de aparelhos com sistemas operacionais diferentes possam fazer requisições a um ou mais servidores. Para que essa comunicação seja possível é necessário os *SOs* usarem os mesmos protocolos.

Conforme descrito por Kurose e Ross (2013) "A Internet é uma rede de computadores que interconecta centenas de milhões de dispositivos de computação ao redor do mundo. Há pouco tempo, esses dispositivos eram basicamente PCs de mesa, estações de trabalho Linux, e os assim chamados servidores que armazenam e transmitem informações, como páginas da Web e mensagens de e-mail. No entanto, cada vez mais sistemas finais modernos da Internet, como TVs, laptops, consoles para jogos, telefones celulares, webcams, automóveis, dispositivos de sensoriamento ambiental, quadros de imagens, e sistemas internos elétricos e de segurança, estão sendo conectados à rede."(KUROSE, 2013)

2.2 Como funciona o protocolo ARP (Address Resolution Protocol)?

Embora na internet cada máquina tenha um ou mais endereços IP, esse endereço não é usado para transmissão de pacotes, devido os aparelhos que fazem o enlace não reconhecerem esse tipo de protocolo como identificador. Para isso é necessário usar o endereço físico. A forma de descobrir o endereço físico tendo apenas o IP é enviando uma uma requisição *ARP* no *Broadcast* para que todos aparelhos recebam a solicitação e apenas o dono do IP responda com o endereço físico usando um ARP reply (IBM , 2023).

2.3 Como funciona o protocolo IP (Internet Protocol)?

Na Internet, cada *host* e cada roteador tem um endereço IP que codifica seu número de rede e seu número de *host*. A combinação é exclusiva: em princípio, duas máquinas na Internet nunca têm o mesmo endereço IP. Todos os endereços IP têm

32 bits e são usados nos campos *Source address* e Destination address dos pacotes IP. É importante observar que um endereço IP não se refere realmente a um *host*. Na verdade, ele se refere a uma interface de rede; assim, se um *host* estiver em duas redes, ele precisará ter dois endereços IP. Porém, na prática, a maioria dos hosts está em uma única rede e, portanto, só tem um endereço IP. (TANENBAUM, 2011)

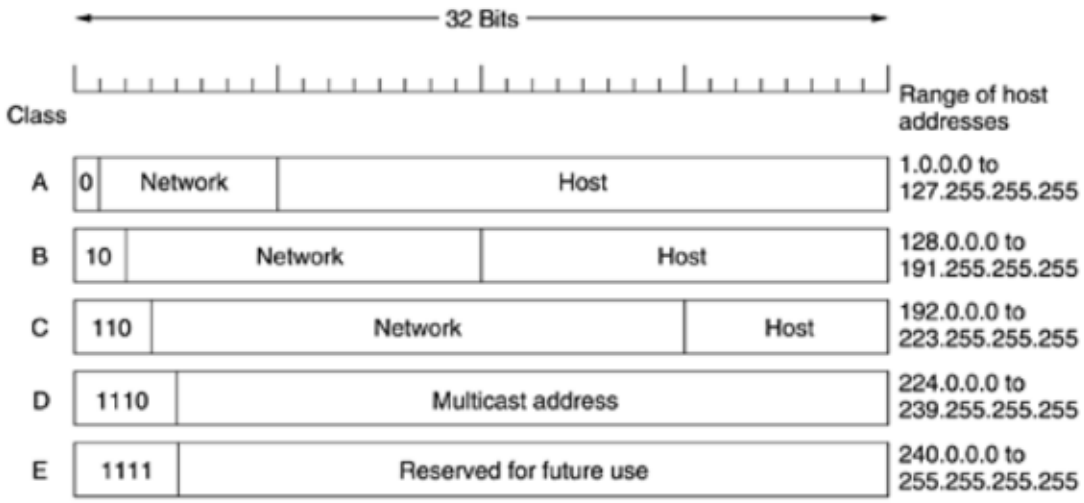
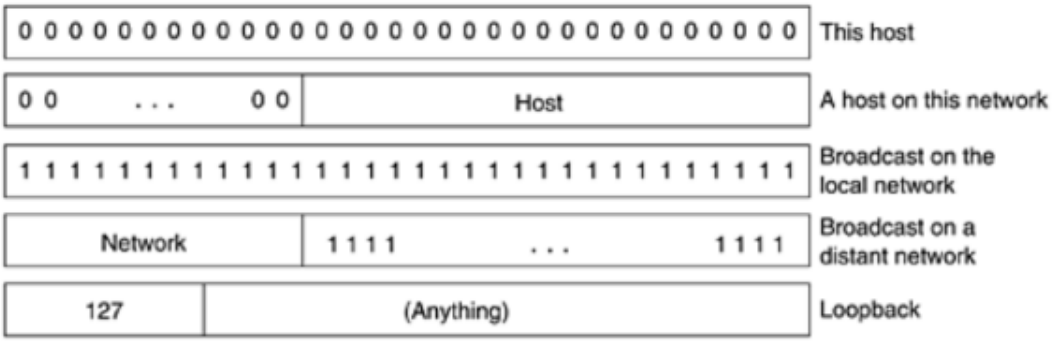


Figura 2 – Formados de endereços IP

Extraído de Tanenbaum (2011)

Descrição do formato de endereços IP para classes dentro de diferentes tipos de sub-rede, como exemplificação da fração de bits dedicados a rede e *hosts*.



Extraído de Tanenbaum (2011)

Endereços IP reservados para casos especiais como envio em *broadcast*, endereço de *loopback*, *host* dentro da mesma rede e o próprio *host*

Em geral, os endereços de rede, que são números de 32 bits, são escritos em notação decimal com pontos. Nesse formato, cada um dos 4 bytes é escrito em notação decimal, de 0 a 255. Por exemplo, o endereço hexadecimal de 32 bits C0290614 é escrito como 192.41.6.20. O endereço IP mais baixo é 0.0.0.0 e o mais alto é 255.255.255.255. (TANENBAUM, 2011)

Como funciona a sub-rede:

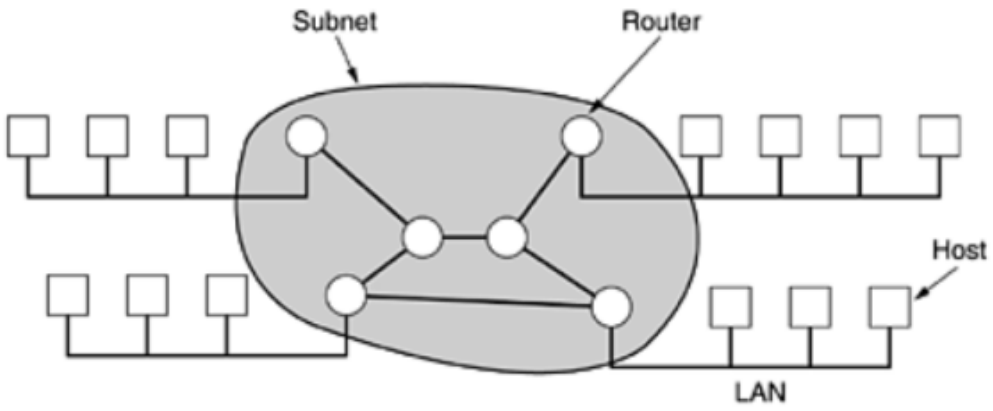


Figura 3 – Relação entre hosts em lan e a sub-rede

Fonte: Tanenbaum (2003), P. 31

Separação física de *hosts* em 4 redes *lan* separadas por 4 sub-redes distintas

2.4 O que são expressões regulares?

Regex vem da abreviação do inglês Regular *expressions*, que por sua vez traduzimos para expressões regulares. Estas são técnicas consideradas de pesquisa e validação dentro de qualquer padrão de caracteres em um texto. Através destas expressões conseguimos buscar alguma sequência específica presente dentro de um texto (ou não!), podemos validar padrões como telefones ou e-mails e senhas. E até mesmo fazer as substituições necessárias dentro desses textos.

"Uma expressão regular é um método formal de se especificar um padrão de texto. Mais detalhadamente, é uma composição de símbolos, caracteres com funções especiais, que, agrupados entre si e com caracteres literais, formam uma sequência, uma expressão. Essa expressão é interpretada como uma regra que indicará sucesso se uma entrada de dados qualquer casar com essa regra, ou seja, obedecer exatamente a todas as suas condições."(JARGAS, 2012)

Uma regex é feita utilizando uma receita, isso mesmo, e para entender esta receita é preciso conhecer primeiro os ingredientes. A expressão regular é tudo aquilo

Figura 4 – Um exemplo de expressão regular

Fonte: Produção própria

presente dentro destas duas “/” ”/”, e o que estiver entre elas é uma forma de pesquisar o que estamos procurando. Porém, para atingir o que buscamos, alguns detalhes são importantes.

São utilizadas “flags” para comandos extras das nossas expressões regulares, segue abaixo algumas delas:

g : Com essa flag, a expressão retorna todas as correspondências, não se limitando a primeira que achar;

i : Aplicando essa flag você está dizendo para expressão que pode ignorar maiúsculas e minúsculas;

m : Com essa flag, a expressão trata caracteres de início e fim (̂$) delimitando a cada nova linha, e não o texto por completo;

Estas flags ficam expostas logo após as barras (“/”) da regex, como mostrado na primeira imagem, como exemplo a expressão /e/gi.

Mas ainda assim ainda existem especificações para compreendermos melhor.

Dentro de uma regex existem dois tipos de caracteres:

Caracteres simples: são os caracteres que já conhecemos, como a, B, c, Z, 1, 2, 3. Que compõem aqueles utilizados em nossa escrita literal, por isso se quisermos pesquisar por algo mais específico é necessário informar para nossa regex da forma que buscamos. abc, 123, abc123, A, a, ã, é, . . .

Existem meta caracteres que funcionam melhor em uma pesquisa, são alguns deles:

| (pipe): funciona como o operador OU, ou seja, a expressão /e|é/ nos traz como resultado a primeira letra E ou É encontrada.

. (ponto): o ponto possui um tratamento diferente, pois naquela posição traz como resultado todo e qualquer caracter. Assim sendo, a expressão /./gi, contendo estas duas flags (g e i) nos resulta com todos os caracteres de entrada, incluindo espaços.

[ ] (conjuntos): podem ser definidos dentro de conjuntos os diversos caracteres

que queremos que sejam encontrados, por exemplo, se quisermos encontrar todas as vogais presentes em um texto, informamos uma expressão semelhante a esta /[aeiou]/gi. Conjuntos também podem ser usados com intervalos de caracteres, por exemplo, quando queremos todos os números de nosso texto podemos fazer algo assim /[0-9]/g. Alguns outros exemplos de conjuntos... [a-z], [A-Z], [À-ú].

Pode ser negado um conjunto através de /[̂0-9]/g, através desta expressão, conseguimos todos os caracteres que não são números de nosso texto.

Também conseguimos fazer com que os meta caracteres possam ser tratados como caracteres simples através de “\”, então "/\ ./"esta expressão nos retorna um único ponto, caso seja encontrado dentro do texto em análise.

Existem muitas outras especificações para uma expressão regular mas não vem ao caso pois este não é o foco desta fundamentação.

2.5 O que é TCP/IP?

O TCP/IP é composto por dois protocolos da camada de transporte, o protocolo TCP (Protocolo de Controle de Transmissão) e IP (Internet Protocol). Estes protocolos juntos tem como objetivo trabalhar a comunicação entre os diversos dispositivos dentro da internet, ou seja, eles especificam como a comunicação deve ser feita dentro da rede, garantindo que a informação sendo transferida chegue ao seu destino de forma correta e segura.

O TCP tem a função de fazer a comunicação entre aplicações através da rede, fazendo com que a mensagem a ser enviada seja quebrada em diversos pedaços e que será reconstruída antes de chegar no seu destino final. Como dito no artigo do (Tecnoblog, 2022) por Ricardo Syozi: “Uma boa analogia é compará-lo a alguém que recebe as peças e, em seguida, monta o quebra-cabeça por completo.”

E por sua vez, o IP é o que define o endereçamento e o caminho dos dados, garantindo que a informação chegue correta ao seu destino. Como dito por (TANENBAUM, 2011) o modelo TCP/IP também conhecido como modelo de referência TCP/IP foi uma nova arquitetura para a comunicação dos hosts, devido a demanda do Departamento de Defesa dos EUA, visando conseguir se manter ativa caso de sua infraestrutura perdesse diversos *hosts*, roteadores e *gateways* de interconexão.

Pode se concluir que o protocolo TCP/IP serve para viabilizar a comunicação entre Origem e Destino, de forma a manter a segurança e integridade dos dados transferidos.

2.6 Diferença de TCP e UDP

A principal diferença entre estes dois protocolos está na integridade. Enquanto o protocolo UDP (User Datagram Protocol) é conhecido por sua falta de confiabilidade, isso quer dizer que através desse protocolo não há como ter a garantia de que os dados enviados chegarão intactos e em ordem correta. Isso porque o UDP não necessita de uma validação para estabelecer comunicação entre as partes.

Ao contrário do UDP, o protocolo TCP (*Transmission Control Protocol*) é conhecido e muito utilizado por sua grande confiabilidade em sua integridade de entrega dos dados. Como explica Yan Orestes no artigo para o site Alura, o protocolo TCP possui uma verificação de três etapas chamada *Three way handshake*, onde para estabelecer uma conexão entre dois hosts, é enviado primeiro um pacote de sincronização, em seguida, o segundo host recebe e responde com outro pacote de sincronização, então, o primeiro responde com um pacote de confirmação para a conexão ser feita.

De maneira mais geral, a rede TCP/IP — disponibiliza dois protocolos de transporte distintos para a camada de aplicação. Um deles é o UDP (User Datagram Protocol — Protocolo de Datagrama de Usuário), que oferece à aplicação solicitante um serviço não confiável, não orientado para conexão. O segundo é o TCP (Transmission Control Protocol — Protocolo de Controle de Transmissão), que oferece à aplicação solicitante um serviço confiável, orientado para conexão. Ao projetar uma aplicação de rede, o criador da aplicação deve especificar um desses dois protocolos de transporte. (KUROSE, 2013)

2.7 Protocolo ICMP

O ICMP (Internet Control Message Protocol) é um protocolo da camada de rede responsável por gerenciar o tráfego de dados e o controle de erros nessa camada. Segundo Kurose (KUROSE, 2013), O ICMP, especificado no [RFC 792], é usado por hospedeiros e roteadores para comunicar informações de camada de rede entre si. A utilização mais comum do ICMP é para comunicação de erros. Por exemplo, ao rodar uma sessão Telnet, FTP ou HTTP, é possível que você já tenha encontrado uma mensagem de erro como “Rede de destino inalcançável”. Essa mensagem teve sua origem no ICMP.", sendo assim, podemos concluir que o ICMP tem a função de relatar erros presentes na camada de rede e diagnosticar esses problemas. Ainda assim, existem atores mal-intencionados que conseguem fazer a manipulação dos recursos do protocolo para negação de serviço (DDoS).

Os roteadores utilizam do ICMP para o gerenciamento da rede, quando a rede proíbe a entrega de um pacote ao seu destino, é enviada uma mensagem de erro para

o dispositivo que a enviou, essas mensagens possuem especificações contendo mais informações sobre o erro que foi causado.

2.8 Portas de comunicação de rede

As portas de rede servem como um ponto virtual para comunicação entre processos, elas determinam onde começam e terminam as conexões de rede. Estas portas são gerenciadas pelo sistema operacional do computador, com um número atribuído a cada porta e a maioria das portas são padronizadas para todos os dispositivos conectados à rede. Elas permitem que os computadores possam diferenciar facilmente entre os tipos de tráfego, os e-mails utilizam uma porta diferente daquela dos serviços web, mesmo que ambas cheguem ao mesmo computador pela mesma conexão de internet. (Cloudflare , 2022)

2.8.1 Testes de penetração (Pen Test)

O objetivo do teste de penetração busca rastrear pontos fracos de segurança em um sistema, rede ou aplicativo. Os testes de segurança nessa área se constroem de forma a simular um ataque real ao alvo, permitindo que os administradores de segurança identifiquem e corrijam as fraquezas antes que um invasor mal-intencionado as usufrua dela. (WEIDMAN, 2014)

De acordo com (MORENO, 2016) "O pentest é uma bateria de testes metodológicos que tem como objetivo descobrir, mapear e expor todas as possíveis vulnerabilidades de uma rede. Há diversos tipos de teste que podem ser realizados: rede cabeada, redes sem fio (wireless), web, revisão do código-fonte, desenvolvimento de programas que exploram vulnerabilidades em outros softwares (exploits) etc."

Um teste de penetração pode ser feito de diversas formas e intensidades, buscando tipos de vulnerabilidades específicas já identificadas anteriormente ou apenas executando uma varredura geral de possíveis oportunidades para uma invasão ou extração de dados valiosos na rede;

2.8.2 Varredura de Portas (Port Scanner)

Como descrito por Mitnick e Simon (2003), no livro - "Hacking - Segredos e Confissões de um Hacker", a principal técnica. A principal técnica de varredura de portas é usando um Port Scanner, que vai rastrear as portas abertas em um dispositivo. Um exemplo disso, é a ferramenta NMAP (PyPi , 2021) muito utilizada por analistas de segurança para testes de segurança em redes. Como a maioria dos sistemas operacionais tem um conjunto de portas padrão dedicadas a serviços específicos. Como serviço de HTTP sendo executado na porta 80, FTP usando a porta 21 e Telnet na porta 23. Um Port Scanner identifica as portas abertas rapidamente, que indica que o serviço pode estar sendo executado naquele dispositivo. (Moreno, 2019)

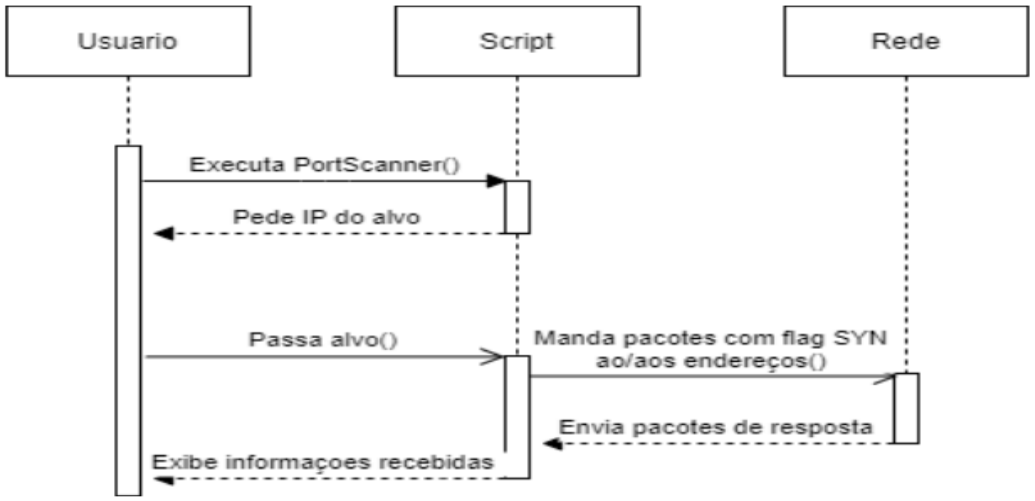


Figura 5 – Diagrama representando os passos do *Postscanner*

Fonte: Produção própria

Diagrama de sequencia mostrando a sequencia de ações para ser executado o *script portscanner*. Iniciando com uma solicitação do destino e logo em seguida iniciando a execução do *Scan* que ao final exibe as informações na tela.

A varredura de portas é essencial em qualquer teste de penetração, já que é a técnica inicial em para identificação de vulnerabilidades de alto risco, a varredura analisa todas as aplicações conectadas com a rede e que fazem sua comunicação com a internet, ou seja, qualquer aplicativo que envie ou receba informações da internet está conectado a uma porta, e qualquer vulnerabilidade nessas aplicações pode causar um grande dano à segurança da rede, já pode possibilitar um possível acesso ou interceptação de pacotes durante o tráfego de informação.

Uma varredura pode ser executada em todas as portas de conexão ou ser personalizada para ler somente portas selecionadas, sejam elas únicas ou em conjunto pré definido (range), aplicações de grande uso comumente usam portas padrões, fazendo com que saibamos o que está sendo executado na porta analisada em qualquer conexão. Como por exemplo:

Porta	Descrição	Protocolo
20	Transferência de dados via FTP	FTP
21	Controle de comando FTP	FTP
22	Login SSH (Secure Shell)	SSH
25	Recebimento de e-mails via SMTP	SMTP
53	Serviço de DNS (Sistema de Nomes de Domínio)	DNS
80	Transferências HTTP	HTTP
110	Recebimento de e-mails via POP3	POP3
143	Recebimento de e-mails via IMAP	IMAP
443	Transferências HTTPS (via TLS/SSL)	HTTPS

Figura 6 – Tabela com algumas das portas conhecidas

Fonte: Produção própria

Algumas das mais comuns portas que são usadas em muitos aplicativos e serviços de rede, junto do protocolo utilizado e aplicação que são frequentemente encontradas.

2.8.3 O que é *Sniffer* ou análise de pacotes?

O *Sniffer* ou analisador de pacote é um método de captura de dados brutos em redes locais ou via Wifi, o *script* atua entre os *Hosts* e o roteador para interceptar a comunicação entre os pontos e assim obter algumas informações sobre o que está sendo enviado na rede analisada.

"*Sniffers* são programas que conseguem capturar (local e/ou remotamente) o tráfego de dados da rede. Como as redes sem fio enviam os dados pelo ar (não sendo

direcionado somente ao destino correto e sim a todos os que estiverem ao redor), qualquer máquina executando um *sniffer* (mesmo um *sniffer* que capture somente os dados direcionados à máquina local, como é o caso do *Wireshark*) consegue fazer a captura remota dos dados". (MORENO, 2016)

Os *Sniffers* são usados frequentemente por hackers ou testadores de penetração para tentar capturar informações críticas de pessoas e empresas, como suas senhas ou arquivos confidenciais, vendo isso, os *Scripts* foram executados em situações que simulam esse tipo de atividade e buscaram obter informações importantes que estavam circulando em nossa rede de testes, mostrando assim a efetividade prática do sistema.

3 TRABALHOS CORRELATOS

Existem vários trabalhos correlatos ao tema de Segurança em Redes Domésticas que podem ser mencionados. Um trabalho importante é o de Ferreira et al. (2017), que propõe um modelo de segurança para redes domésticas que integra diferentes tecnologias, como firewalls, antivírus, criptografia e monitoramento de rede. O modelo proposto foi avaliado por meio de testes de desempenho e demonstrou ser eficaz na proteção da rede doméstica.

Outro trabalho relevante é o de Silva et al. (2018), que apresenta uma abordagem para detecção de ataques em redes domésticas por meio de técnicas de machine learning. O trabalho utiliza algoritmos de classificação para identificar atividades maliciosas na rede, como escaneamento de portas e ataques de força bruta. Os resultados experimentais mostraram que a abordagem proposta é capaz de identificar ataques com alta precisão.

Além disso, há trabalhos que se concentram em tecnologias específicas de segurança em redes domésticas, como o trabalho de Han et al. (2019), que apresenta uma solução de firewall baseada em aprendizado de máquina para redes domésticas. O firewall proposto é capaz de detectar e bloquear ataques de forma automática, sem a necessidade de intervenção do usuário.

Por fim, é possível mencionar trabalhos que exploram vulnerabilidades específicas em dispositivos e protocolos de rede utilizados em redes domésticas, como o trabalho de Garcia et al. (2016), que investiga vulnerabilidades em dispositivos IoT utilizados em redes domésticas. O trabalho identifica várias vulnerabilidades em dispositivos como câmeras de segurança, roteadores e dispositivos de armazenamento em nuvem, destacando a importância de medidas de segurança adequadas em redes domésticas.

Trabalhos	Funcionamento
Garcia et al. (2016)	Investiga vulnerabilidades em dispositivos IoT utilizados em redes domésticas. detectando vulnerabilidades em dispositivos como câmeras de segurança, roteadores e dispositivos de armazenamento em nuvem.
Ferreira et al. (2017)	Desenvolveu e integrou tecnologias de proteção de rede como Firewalls, antivírus, criptografia e monitoramento de rede.
Silva et al. (2018)	Detecta ataques em redes domésticas por meio de técnicas de machine learning.

Figura 7 – Tabela de trabalhos correlatos

Fonte:Produção própria

4 DESENVOLVIMENTO

4.1 Metodologia

Para o desenvolvimento do presente trabalho, o primeiro passo após a escolha do tema, que aborda a segurança de dados nas redes de computadores, foi decidir qual linguagem seria usada para a criação do *script* e quais seriam as funções de análise de rede que seriam implementadas no sistema. Após uma pesquisa feita em busca de encontrar a melhor linguagem e funções, foi decidido que a melhor opção seria o Python devido a facilidade proporcionada em trabalhar com redes apenas com algumas bibliotecas importadas, reduzindo assim a verbosidade e aumentando a compreensão do código e liberdade para alterações significativas sem necessitar de uma reformulação de total.

Uma vez decidida a linguagem foram iniciados os estudos nos livros sobre o assunto para se obter exemplos de construção de *Sniffers* e *Port Scanners*, com o intuito de fazer um apanhado de ideias e depois incluir as ideias vistas nos estudos em um protótipo real e funcional.

Iniciamos o projeto, com o levantamento bibliográfico que foi realizado através de diferentes fontes, como: artigos, livros e sites, visando aprofundar o conhecimento sobre o tema proposto e entender quais as principais ferramentas para um *Pentest* efetivo.

Para não fazer as análises em locais como praças que possuem uma infinidade de variáveis que podem deixar o trabalho com análises inconclusivas e aumentar o controle foram usada redes domésticas com diferentes combinações de aparelhos conectados, como televisores, celulares, computadores e até o roteador de internet.

O caráter contido no projeto é de pesquisa exploratória, já que o desenvolvimento do *script* foi feito visando o mapeamento de portas de comunicação entre aplicações e a possibilidade de analisar os dispositivos conectados em uma mesma rede, fazendo assim com que sejam coletados diversos dados para análise de segurança das redes e colaborar na proteção de dados pessoais que possam estar em risco devido a conexões de baixa segurança.

4.2 Desenvolvimento

Esse capítulo tratará da descrição dos passos durante o desenvolvimento dos *scripts* para *Pentest*.

4.3 Tecnologias Envolvidas

4.3.1 Linguagem Python

Os Scripts foram desenvolvidos usando a linguagem Python, sendo ela uma linguagem de tipagem forte e dinâmica, sendo o Python a linguagem mais utilizada na programação no momento, muitas documentações, bibliotecas e projetos da comunidade estão disponíveis para estudo, facilitando todo o processo de desenvolvimento dos *Scripts* e também a qualidade, já que diversas funções usadas foram escolhidas de acordo com as documentações da linguagem disponíveis.

Além de ser uma linguagem de alto nível e multi-paradigma, o Python é uma das melhores opções no quesito segurança e rede, já que oferece soluções nativas para isso e também já foi muito aplicada em projetos semelhantes, garantindo assim o funcionamento efetivo e seguro da execução dos *Scripts* em qualquer rede.(PYTHON, 2023)

4.3.1.1 Biblioteca NMAP

Para facilitar nas consultas e obtenção de informações foi usada a biblioteca NMAP que fornece uma ampla quantidade de funções que permitem usar protocolos de comunicação diferentes, definir a faixa de IP a ser analisada, intensidade da análise entre outras opções. Assim como descrito na própria documentação da biblioteca “python-nmap é uma biblioteca python que ajuda a usar o *scanner* de porta nmap. Ele permite manipular facilmente os resultados de varredura do nmap e será uma ferramenta perfeita para administradores de sistemas que desejam automatizar tarefas e relatórios de varredura. Ela também suporta saídas de *script* nmap.” (PyPi , 2021)

4.3.1.2 Biblioteca Scapy

Para a geração do *Sniffer* foi indispensável a utilização da biblioteca Scapy, disponível também para Python 2. Ela se baseia na ferramenta Scapy que é direcionada a manipulação de pacotes que trafegam na rede, também é muito utilizado para tarefas como varreduras, testes unitários, análises de vulnerabilidades e muito mais que envolve lidar com pacotes de comunicação. Com ela, devido às funções nativas foi possível gerar o interceptador de pacotes sem muitas linhas de código, e devido a

sua fama fica muito mais fácil de localizar material para estudo na hora de fazer a implementação e melhorias. (Scapy, 2022)

4.3.1.3 Biblioteca Matplotlib

Para a demonstração dos resultados obtidos de uma maneira simples e eficiente, foi utilizada a biblioteca Matplotlib, que é responsável por criar gráficos 2D e 3D com base nas informações obtidas no resultado de uma função do código, com essa biblioteca, foi possível transformar as análises e operações resultantes dos *scripts* em algo visualmente interessante e intuitivo para o usuário, criando assim um diferencial na exibição dos resultados quando comparado a outras opções de varredura e farejamento de rede disponíveis no mercado. (PyPi , 2023a)

4.3.1.4 Biblioteca Tabulate

A fim de organizar os resultados obtidos em nossos *scripts*, usamos algumas bibliotecas do Python para otimizar a exibição, entre elas, a biblioteca Tabulate. Esta biblioteca é responsável por organizar as saídas em uma linha de comando de forma que possibilite a organização das informações em formato de colunas e linhas, assim como uma tabela, simplificando assim a organização das informações da forma mais intuitiva possível. (PyPi , 2022)

4.3.1.5 IpAdress

Para validação dos IPs fornecidos pelos usuários, foi utilizada a biblioteca de validação IpAdress, que permite manipular os endereços virtuais com facilidade e eficiência. Citação: "ipaddress fornece recursos para criar, manipular e operar em endereços e redes IPv4 e IPv6."(Moody, P , 2023)

4.3.1.6 Biblioteca Rich

No código, a biblioteca foi usada para dar ênfase nas informações de maior importância e facilitar a compreensão das informações apresentadas no terminal. Evitando que informações de grande importância na fase de análise dos resultados passem despercebidas. Citação: "Rich é uma biblioteca Python para escrever rich text (com cor e estilo) no terminal e para exibir conteúdo avançado, como tabelas, markdown e código destacado de sintaxe."(McGugan, W , 2023)

4.3.1.7 Biblioteca TermColor

Em conjunto a biblioteca apresentada anteriormente, a biblioteca TermColor funciona em conjunto com a Tabulate, fazendo com que as principais informações obtidas sejam destacadas em cores especificas, facilitando ao usuário o acesso as

informações mais importantes durante o processo de testes de penetração. (PyPi , 2023b)

4.3.1.8 Biblioteca OS

A biblioteca OS foi utilizada para fazer automação de processos no Python, usando funções de sistemas operacionais para realizar operações dentro do computador e tornando o processo de desenvolvimento mais rápido e prático. (Python Documentation , 2022)

4.3.1.9 Biblioteca Time

Esta biblioteca, como o próprio nome já diz, é responsável por resultar em uma data a partir de um ponto fixo no tempo que é chamado de "era", ponto este que varia entre plataformas. Sendo assim limitado a somente datas posteriores a esta "era". Neste trabalho, a biblioteca Time foi utilizada para nos trazer a hora específica da leitura de uma entrada no momento da saída. (Python Documentation , 2023b)

4.3.1.10 Biblioteca DateTime

Como uma complementação de informações para a biblioteca anterior, este módulo vem com o intuito de trazer informações adicionais de data e hora para a saída desejada com a possibilidade de conversão de fuso horário. (Python Documentation , 2023a)

4.3.2 Sistema operacional Linux

A escolha de um sistema operacional para o desenvolvimento dos *Scripts* foi essencial para que o projeto pudesse ser feito, já que muitas funções nativas da distribuição Parrot OS do Linux foram usadas para que se entendesse o funcionamento e execução das leituras de rede e interceptação das informações nela.

O Parrot OS (Parrot, 2023) é uma distribuição Linux baseada em Debian (Debian, 2023). É um sistema operacional meticulosamente elaborado que atende especificamente a analistas de rede e testadores de penetração. A presença de uma infinidade de ferramentas que vêm pré-instaladas com o Parrot OS o transforma em uma das opções mais escolhida para profissionais do meio da segurança atualmente, sendo uma das opções com versões mais recentes para este propósito disponível no mercado. O uso deste sistema nos possibilitou realizar leituras e comparações de maneira rápida, podendo assim ser usado de base para os *Scripts* e suas funções atendidas.

4.3.3 Equipamentos e Hardware

Para os testes de execução e análises de desempenho dos *Scripts*, foi necessário o uso de diversos dispositivos com configurações de hardware diferentes, visto que a capacidade de processamento de cada dispositivo pode impactar na velocidade de execução do *Script*, o que é um fator muito importante para sua efetividade no uso proposto. Com intuito de analisar uma rede Wifi pública, foi utilizado um receptor de redes WI-FI de alta potência TP-LInk TL-WN822n, esse dispositivo foi essencial no desenvolvimento pois tornou possível fazer a leitura de pacotes sem a necessidade de estar conectado fisicamente ao host que deseja monitorar, que com apenas uma antena externa seria possível ter essa flexibilidade, pois placas nativas de notebooks e até algumas de *desktops* vem com bloqueio de fabrica para leitura de tráfego de outros hosts.

Figura 8 – Receptor wi-fi TL-WN822N

Fonte: (TP-Link , 2023)

Imagem de receptor Wi-fi usado durante os testes e simulações de rede.

O dispositivo denominado 'TL-WN822N' foi empregado com o intuito de otimizar o desempenho e ampliar o alcance na análise do tráfego de pacotes dos *hosts* que se encontram envolvidos em comunicação sem fio. Tal implementação elimina a imperatividade de estabelecer uma conexão física direta com o dispositivo que se deseja monitorar, proporcionando uma abordagem mais eficiente e flexível no acompanhamento do tráfego.

4.3.4 Sniffer Wireshark

Neste trabalho foi usado para ler os pacotes enviados pelo *Port Scanner* em tempo de execução. Sua escolha se dá por ser a ferramenta mais completa encontrada, ter capacidade de exibir os pacotes em tempo de execução, aplicar filtros de exibição e devido a vir já instalado em diversas distribuições Linux voltadas a segurança digi-

tal como Kali e Parrot OS simulando assim mais realisticamente um cenário real e descartando a instalação de ferramentas adicionais. (Wireshark ,2023).

4.4 Análise e tratamento de resultados

Os resultados analisados na execução dos *Scripts* seguiram duas vertentes, sendo elas a velocidade de execução e os dados e informações capturados na rede, ambos resultados trabalham em conjunto para a maior eficiência dos *Scripts*, já que a execução é feita em uma rede online que transmite dados em tempo real.

4.4.1 Velocidade de execução

A velocidade de execução foi um ponto chave para o desenvolvimento do projeto, já que para ser efetivo deveria fazer a varredura de rede e a tentativa de Sniffing o mais rápido possível, olhando para outros exemplos de *Scripts* de varredura de rede, conseguimos identificar os principais pontos para que o *Script* fosse executado em uma rede real com diversos usuários sem atrasos desnecessários. Para que isso fosse possível, foram feitas diversas alterações que buscaram “polir” o *Script* de maneira que fosse eficiente em máquinas com pouco poder de processamento e não requeresse muito esforço computacional para analisar redes com usuários reais conectados e a alta troca de dados.

Nas imagens a seguir podemos ver a diferença de velocidade de execução dos *Scripts* no escaneamento de portas e mapeamento da rede em dois dispositivos de configurações diferentes e usando três modos de execução, sendo analisadas as situações de: Rede local, Rede em um hospedeiro e Rede com 8 hospedeiros.

ANALISE DE DESEMPENHO - PORTSCANNER			
		TEMPO MEDIO(3x)	
VM Ryzen 5 4 nucleos 3500 Mb RAM	Analise em localhost	Tempo medio de analise em range de 20 a 5800	1,0 segundos
		Tempo medio de analise em portas famosas	0,57 segundos
	Analise em um host	Tempo medio de analise em range de 20 a 5800	11,52 segundos
		Tempo medio de analise em portas famosas	10,72 segundos
	Analise em rede local de classe C(8 hosts)	Tempo medio de analise em range de 20 a 5800	57,14 Segundos
		Tempo medio de analise em portas famosas	27,52 Segundos
		TEMPO MEDIO(3x)	
VM Ryzen 5 2 nucleos 2000 Mb RAM	Analise em localhost	Tempo medio de analise em range de 20 a 5800	1,03 segundos
		Tempo medio de analise em portas famosas	0,64 segundos
	Analise em host	Tempo medio de analise em range de 20 a 5800	12,38 segundos
		Tempo medio de analise em portas famosas	11,99 segundos
	Analise em rede local de classe C(8 hosts)	Tempo medio de analise em range de 20 a 5800	69,47 Segundos
		Tempo medio de analise em portas famosas	40,18 Segundos

Figura 9 – Testes de desempenho de velocidade

Fonte:Produção própria

Na Figura 10 podemos ver os diferentes tempos para escanear os hosts de uma rede em duas condições de maquina e três modos de leitura, com isso ficou claro

que a melhora de processador e RAM da máquina fica mais perceptível à medida que análise seja mais ampla nos parâmetros de hosts e portas. Os dados em em forma de dicionário para o python, com as chaves e valores sendo separadas e exibidas posteriormente no código

Os resultados obtidos nos testes durante o desenvolvimento foram cruciais para que pudéssemos chegar ao resultado atual, já que a cada mudança no *Script* era diretamente ligada a velocidade de sua execução e sua efetividade prática. Olhando também para os resultados podemos entender a maneira que a análise pode ser configurada de maneira efetiva pelo usuário, visto que os resultados de velocidade se mostraram melhores em redes analisadas por completo ao invés de análises únicas para cada *Host*.

4.4.2 Resultados obtidos

Os resultados dos *Scripts* foram divididos também em duas vertentes, sendo os resultados do mapeamento das portas de conexão dos dispositivos conectados à rede em que os testes foram executados e também a tentativa de captura de informações e dados que trafegam entre os *hosts* e o roteador na rede Wifi.

O status de segurança da rede analisada é dado de acordo com a facilidade e a quantidade de informações obtidas em cada resultado da execução, em redes menos seguras, os *Scripts* são capazes identificar aplicações sendo executadas em portas erradas e capturar dados facilmente por meio do *Sniffer*, isso irá classificar uma rede não segura para se conectar, e o mesmo acontece para redes seguras, onde quanto menos dados forem obtidos na conexão, melhor para o usuário da rede escaneada.

4.4.3 Resultado de mapeamento de redes

O mapeamento de rede foi executado em diversas conexões simuladas com fim de identificar as vulnerabilidades e aplicações que estavam rodando nas portas de conexão do *host* conectado a rede, nestes resultados do *Script* conseguimos coletar informações importantes sobre o tipo de conexão do *Host* e suas aplicações online, como por exemplo:

- Tipo de conexão do *Host*
- IP do usuário
- Nome da máquina conectada à rede
- Porta de conexão aberta
- Tipo da porta de conexão

- Aplicação operando na porta de conexão aberta
- Versão do sistema operacional da máquina do *Host*

Na Figura abaixo podemos ver um exemplo de resultado obtido em uma conexão simulada onde uma porta de conexão estava aberta e vulnerável na rede, possibilitando assim a invasão ou manipulação dos seus dados da rede Wifi em que o usuário estaria conectado, para facilitar a leitura, os dados obtidos por nosso *Port Scanner* são passados a um JSON e separados por sua classe, podendo assim ser lido com mais clareza para a fácil identificação da vulnerabilidade por parte do cliente.


```
{
   "127.0.0.1":{
      "hostnames":[
         {
            "name":"localhost",
            "type":"user"
         },
         {
            "name":"localhost",
            "type":"PTR"
         }
      ],
      "addresses":{
         "ipv4":"127.0.0.1"
      },
      "vendor":{

      },
      "status":{
         "state":"up",
         "reason":"conn-refused"
      },
      "tcp":{
         "3389":{
            "state":"open",
            "reason":"syn-ack",
            "name":"ms-wbt-server",
            "product":"xrdp",
            "version":"",
            "extrainfo":"",
            "conf":"10",
            "cpe":"cpe:/a:jay_sorg:xrdp"
         }
      }
   }
}
```


Figura 10 – Resultados obtidos durante o mapeamento

Fonte: Produção própria

Na figura 11 temos um exemplo de como os dados são fornecidos após a análise da rede com o *port scanner*.

4.4.4 Resultado da captura de pacotes

A captura de pacotes ou *Sniffer* consiste em capturar dados na comunicação da rede, os resultados para a captura de dados são em geral dados brutos da comunicação entre cliente e servidor, onde os dados capturados são “brutos” e ilegíveis, porém, com ferramentas de tratamento de dados podemos realizar um processo de limpeza dos dados para coletar informações de uma forma legível, manipulável e muito mais produtiva, vendo que é transitado na rede, como na figura abaixo:

```
Ether / IP / TCP 192.168.1.17:51368 > 192.168.0.109:7680 S
Ether / IP / TCP 192.168.1.17:51367 > 192.168.0.109:7680 S
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https S
Ether / IP / UDP / DNS Qry "b'ntp.msn.com.'"
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 SA
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https A / Padding
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https PA / Raw
Ether / IP / UDP / DNS Ans "b'www-msn-com.a-0003.a-msedge.net.'"
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 A / Padding
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https A / Padding
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 A / Raw
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 PA / Raw
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https A / Padding
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https PA / Raw
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https PA / Raw
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https PA / Raw
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 A / Padding
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 PA / Raw
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 PA / Raw
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https A / Padding
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 A / Padding
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 PA / Raw
Ether / IP / TCP 204.79.197.203:https > 192.168.1.17:51369 A / Padding
Ether / IP / TCP 192.168.1.17:51369 > 204.79.197.203:https PA / Raw
```

Figura 11 – *Prompt* com resultados da captura de pacotes

Fonte: Produção própria

Como pode ser visto na Figura 11, os pacotes lidos pelo *sniffer* sem tratamento em código são de difícil compreensão e com quase nem uma informação sobre cada pacote.

Na execução acima não foram aplicados nenhum filtro de aparelho ou protocolo, o objetivo nesse ponto era apenas testar a execução do *Script* na rede para depois expandir para as buscas com inserções de filtros como buscar comunicação de aparelhos e filtrar protocolos como ICMP, TCP e UDP. Posteriormente também foi inserido

um limitador de recebimentos, fazendo com que a quantidade de pacotes obtidos na rede seja limitada a um número predeterminado no início da execução, facilitando a aplicação do *Script* em redes com muitos usuários, podendo assim ser alterado de acordo com a necessidade do cliente.

4.4.5 Codificação

4.4.5.1 Port Scanner

```
80  if __name__ == '__main__':
81      while True:
82          start_time = time.time()
83          try:
84              user_inputs = get_user_input()
85              alvoG, portasSelecionadasG = user_inputs
86              start_scan(alvoG, portasSelecionadasG)
87          except KeyboardInterrupt:
88              f_print(formater_text('\nManual finalization performed',
89                                    FontTypes.ERROR))
90          # except BaseException as e:
91          #     print(e)
92
93          elapsed_time = time.time() - start_time
94          if elapsed_time < 2:
95              f_print(formater_text('\n### There was an error processing your entries,'\
96                  'please review and try again ###\n', FontTypes.ERROR))
97          else:
98              break
```

Figura 12 – Código principal *Port Scanner*

Fonte: Produção própria

No trecho de código da Figura 12 pode ser visto que tudo é executado dentro de um grande if que só será executado caso seja chamado o arquivo do *script* diretamente. Após isso é aberto um *loop* que solicita as informações de porta, *hosts* ou *host*. Em seguida é chamada a função "start scan"com com valores informados pelo usuário, dentro dela será executado e escaneamento da rede é exibido os resultados. Também é visível um tratamento de erro para interrupção manual com a tecla ctrl + C e se caso o lapso de tempo de execução do escaneamento for menor que 2 segundos, o sistema exibe um erro de processamento das entradas e solicita a entrada novamente das informações.

```
31      ip_list = []
32      for i in psutil.net_if_addrs().values():
33          for j in i:
34              if j.family == socket.AF_INET:
35                  ip_list.append(j.address)
36      return ip_list
```

Figura 13 – Código de captura de IP

Fonte: Produção própria

Neste código presente na Figura 13, podemos ver a criação de uma lista, ela vai ao final receber os endereços que forem identificados. Em seguida é executado o comando da biblioteca psutil que extrai informações sobre as interfaces de rede do computador e adiciona a lista 'iplist'. Ao final a lista é retornada contendo todos os IPv4 que o código conseguiu identificar no *host* que executou a função.

```
14  def start_scan(target_ip_address:str, selected_ports:str):
15      f_print(formater_text('Scan started, please wait...', FontTypes.BOLD))
16      scanner = nmap.PortScanner()
17
18      start_read_time = time.time()
19      result = scanner.scan(target_ip_address, selected_ports, '-A -sS --osscan-guess --min-rate=50000 -T3')
20      end_read_time = time.time()
21
22      display_results(result)
23      print(f"Processing time: {(end_read_time - start_read_time):.2f}")
24
25      save_to_file(result)
```

Figura 14 – Código de Start para Port Scanner

Fonte: Produção própria

Este trecho de código contido na Figura 14 define uma função chamada start-scan, que realiza uma varredura de portas em um ou mais endereços IPs. Na primeira linha é exibida uma mensagem informando que o escaneamento começou. Logo após isso, o código cria uma instância de objeto 'PortScanner' da biblioteca nmap. Já na linha 18 é armazenado o tempo logo antes de começar a varredura, que inicia na linha 19 e fica salva no objeto 'result'. Na linha 20 é pego o horário atual e armazenado no objeto 'end-red-time' que vai ser subtraído com o horário contido no objeto 'star-read-time' para se obter o tempo de processamento da leitura. Ao final é exibida as informações obtidas por meio da função 'display-result' passando o valor do objeto 'result', exibido o tempo de processamento, é chamada a função 'save-to-file' passando o valor de 'result' para salvar em arquivo de texto a análise.

4.4.5.2 Sniffer

```
print('Enter a local IPV4 address to be analyzed, or leave it blank to analyze the entire local network')
monitored_device = host_ip_input(accept_empty = True)

pcap_file_name = choose_saving_options()
protocol = choose_protocol()
counter = get_packet_count()

t1 = time.time()
reading_filter = protocol
if monitored_device: reading_filter += f'{"" if not protocol else " and "}host {monitored_device}'


print('Reading started...')
print(monitored_device, pcap_file_name)
if pcap_file_name:
    print(1)
    sniff(iface=interface, prn=packet_handler_save, count=counter, store=0,
          filter=reading_filter)
else:
    print(2)
    sniff(iface=interface, prn=packet_handler, count=counter, store=0, filter=reading_filter)

t2 = time.time()
total = t2 - t1
print(f"Time for reading: {total}")
```

Figura 15 – Código principal sistema Sniffer

Fonte: Produção própria

Como pode ser visto na Figura 15, logo ao início um IPv4 para ser monitorado e salvo em ‘monitored-device’, caso não for passado nada o código vai ler a rede toda. Em seguida são chamadas funções que vão solicitar ao usuário informações de salvamento da leitura, protocolo utilizado e quantidade de pacotes a serem armazenados. Um pouco depois já é pego o horário atual e armazenado em ‘t1’. Depois disso é executado uma validação, que se o usuário tiver informado um host e protocolo de filtro vai concatenar tudo em uma string que vai ser usada como filtragem do sniffer. Após isso é exibido que foi iniciada a leitura. Caso o usuário tenha informado nome para salvamento, é executada a função ‘sniff’ da biblioteca Scapy passando o parâmetro ‘prn’ igual a função packet-handler-save, que é montada para salvar os pacotes em arquivo pcap, caso não tenha sido passado nome para salvamento a função ‘sniff’ executa com o parâmetro ‘prn’ contendo a função ‘packet-handler’ que apenas exibe a leitura no terminal. Ao final é parado de contar o tempo de execução, subtraído o final do inicial contido em ‘t1’ e exibido por meio da função print.

Código de escolha de protocolo

```
107     protocol = input('Would you like to filter a protocol? (Y/N) ')
108     if protocol.lower() == "s":
109         while True:
110             protocol_type = input("Which protocol would you like to filter? 1 = ICMP || 2 = TCP || 3 = UDP ")
111             if protocol_type in ['1', '2', '3']:
112                 if protocol_type == '1':
113                     return 'icmp'
114                 elif protocol_type == '2':
115                     return 'tcp'
116                 else:
117                     return 'udp'
118             else:
119                 f_print(formater_text('Invalid protocol. Please enter 1, 2 or 3.', FontTypes.ALERT))
120     elif protocol.lower() == 'n':
121         return ''
122     else:
123         raise ValueError('Input is not Y or N')
```

Figura 16 – Código de escolha de protocolo Sniffer

Fonte: Produção própria

O código exibido na Figura 16 começa com o terminal solicitando se usuário deseja filtrar algum protocolo. Caso seja informado que sim é iniciado um loop que vai solicitar ao usuário qual protocolo deseja filtrar, podendo ser TCP, UDP ou ICMP. Caso o usuário opte por não filtrar os pacotes o código retorna uma string vazia.

```
f_print(formater_text("*******PACKAGE SUMMARY*******",
                      FontTypes.BOLD))
print(packet.summary())
print()
f_print(formater_text("*******PACKAGE DETAILS*******",
                      FontTypes.BOLD))
print(packet.show())
f_print(formater_text("*******PACKAGE END*******\n\n",
                      FontTypes.BOLD))
```

Figura 17 – Código de análise de pacotes únicos

Fonte: Produção própria

Na Figura 17 é mostrado o trecho de código responsável por exibir no terminal as informações lidas pela função ‘sniff’. Com alguns tratamentos para facilitar a compreensão de onde começa e termina o pacote, além de suas informações em resumo com a função ‘packet.summary’ e sem detalhes com a função ‘packet.show’.

4.4.5.3 Monitor

```
46  def packet_handler(packet):
47      global received_counter
48      if packet.haslayer(IP) and packet[IP].src != monitored_ip:
49          global send_counter_by_source
50          send_counter_by_source.append(packet)
51      if packet.haslayer(IP) and packet[IP].dst == monitored_ip:
52          received_counter += 1
```

Figura 18 – Código de função monitor

Fonte: Produção própria

Como pode ser visto na Figura 18, a função 'packet-handler' puxa o objeto received-counter que é responsável por contar quantos pacotes foram lidos para o mesmo destino. Em seguida é verificado se o IP de origem é diferente do IP que está sendo monitorado, caso seja diferente o código puxa o objeto 'send-counter-by-source' e adiciona a ele o pacote atual. Também é verificado na linha 51 se o IP de destino do pacote é o IP que foi selecionado para ser monitorado, em caso positivo o contador de recebidos puxado no início da função é incrementado em 1.

```
for pkt in send_counter_by_source:
    if pkt.haslayer(IP):
        ip_src = pkt[IP].src
        if ip_src in hosts:
            hosts[ip_src] += 1
        else:
            hosts[ip_src] = 1

for hosts, count in hosts.items():
    if count > maximum_to_alert:
        source_list.append(hosts)
```

Figura 19 – Código de validação de pacotes suspeitos

Fonte: Produção própria

A Figura 19 mostra o trecho de código que faz a atribuição de um contador de pacotes enviados pela mesma origem, onde 'send-counter-by-source' é a lista de pacotes, que em seguida cada pacote dentro dessa lista é analisado e incrementado 1 a cada vez que uma origem é lida. No segundo for é verificado cada a quantidade de

vezes que uma origem foi detectada, caso seja maior que o máximo permitido o IPv4 de origem é adicionado à lista ‘souce-list’ que são os hosts considerados suspeitos.

4.4.5.4 Lançador

```
49  if __name__ == '__main__':
50      try:
51          packet = build_packet()
52
53          print(packet.show())
54          print(packet.summary())
55
56          send(packet)
57      except KeyboardInterrupt:
58          print('')
59          f_print(formater_text('\nManual interruption', FontTypes.ERROR))
```

Figura 20 – Código principal lançador

Fonte: Produção própria

Contido na Figura 20 temos o if que só será executado caso o código seja chamado diretamente pelo python. Nele pode ser visto dentro de um try a chamada da função ‘build-packet’ que é responsável por solicitar as entradas do usuário, como IPv4 de destino, porta, protocolo e demais informações que serão utilizadas pelo código, e o resultado final inserido no objeto ‘packet’. Em seguida é mostrado o pacote em detalhes e seu versão resumida por meio do ‘packet.show’ e ‘packet.summary’ respectivamente. Ao final é chamada a função ‘send’ que envia o pacote montado e caso o código seja interrompido é exibida uma mensagem informando.

4.4.6 Aplicações

4.4.6.1 PortScanner

Como pode ser visto, na Figura 21 o código inicia passando os IPv4 identificados no computador, em seguida solicita ao usuário o alvo da que será escaneado, dando alguns exemplos de como analisar uma rede inteira. O usuário nesse exemplo informou 192.168.1.17 como alvo. Em seguida o terminal solicita ao usuário que passe se deseja informar as portas a serem analisadas, como o caso foi positivo o código solicita que entre com quais portas, podendo ser específicas, um intervalo ou apenas uma, no exemplo o usuário informou o intervalo da porta 0 até a 6000. Caso o usuário opte por não passar as portas, o código vai escanear as portas que são frequentemente utilizadas em servidores e aparelhos responsáveis pela comunicação na rede.

A Figura 22 exibe a saída da execução do Port Scanner no aparelho 192.168.1.17, que como pode ser visto tem diversas portas que responderam o sinal SYN com o sinal

```
Welcome

Your IPv4 are:['127.0.0.1', '192.168.1.9', '172.17.0.1']

        Time to select one or more host(s)
        Provide the subnet using CIDR notation at the end of the IPv4 address if you want to analyze the entire network

        Class legend
        Class A subnet 255.0.0.0 = /8
        Class B subnet 255.255.0.0 = /16
        Class C subnet 255.255.255.0 = /24

Insert the IPv4: 192.168.1.17

Do you want to choose the port(s) to be analyzed? If not,the analysis will select the most frequent service ports
Y to yes
y

                    ### Follow one of the examples ###

                    Single port
                        8080

                    Two or more specific ports
                        443,8080

                    Port range from 0 to 5000
                        0-5000

Enter the destination port: 0-6000
Scan started, please wait...
```

Figura 21 – Terminal de entrada do Port Scanner

Fonte: Produção própria

SYN-ACK, que representa a resposta do host quando se está aberto para conexão. Também é exibido o serviço provável que esteja sendo executado nessa porta, além do endereço MAC do host, Fabricante, tipo da resposta, que no caso foi um 'arp-response', o sistema operacional provavelmente utilizado pelo alvo, tempo de execução e solicitação do nome de arquivo que ficará salva a leitura.

```
***HOST LIST***
Number of devices: 1

ID: 1
IP: 192.168.1.17
Result of search using TCP send
Port found: 135 / Reason - syn-ack
   Service: running in port 135: msrpc
Port found: 139 / Reason - syn-ack
   Service: running in port 139: netbios-ssn
Port found: 445 / Reason - syn-ack
   Service: running in port 445: microsoft-ds
Port found: 5040 / Reason - syn-ack
   Service: running in port 5040: unknown
Port found: 5357 / Reason - syn-ack
   Service: running in port 5357: http
MAC: 48:E7:DA:69:19:F3
Fabricator: AzureWave Technology
Answer type: arp-response
Operational system: Microsoft Windows 10

Processing time: 202.49

Informe o nome do arquivo para salvamento da leitura:
saida em texto
Resultado da analise salvo como 'saida em texto.txt', ele esta salvo na pasta localizada o script
```

Figura 22 – Terminal de saída do Port Scanner

Fonte: Produção própria

A Figura 22 exibe a saída da execução do Port Scanner no aparelho 192.168.1.17, que como pode ser visto tem diversas portas que responderam o sinal SYN com o sinal SYN-ACK, que representa a resposta do host quando se está aberto para conexão. Também é exibido o serviço provável que esteja sendo executado nessa porta, além do endereço MAC do host, Fabricante, tipo da resposta, que no caso foi um 'arp-response', o sistema operacional provavelmente utilizado pelo alvo, tempo de execução e solicitação do nome de arquivo que ficará salva a leitura.

4.4.6.2 Sniffer

```
Enter a local IPV4 address to be analyzed, or leave it blank to analyze the entire local network
Enter the IP address of the destination host: 192.168.1.1
1
Do you want the reading to be saved? (Y/N)y
Would you like to filter a protocol? (Y/N) y
Which protocol would you like to filter? 1 = ICMP || 2 = TCP || 3 = UDP : 1
Enter the number of packages you want to analyze: 10
Reading started...
```

Figura 23 – Terminal de entrada do Sniffer

Fonte: Produção própria

Na Figura 23 temos o terminal de entradas do usuário. Que inicia perguntando se o usuário deseja filtrar a leitura que envolva um *host* específico, ou se deseja exibir todos os pacotes lidos na rede, independente da origem ou destino. Em seguida é perguntado se o usuário deseja salvar a saída. Logo após isso p usuário informa se deseja filtrar algum protocolo, como a resposta foi positiva é passada a opção de *TCP*, *UDP* ou *ICMP*. Caso a resposta tivesse sido negativa o *sniffer* iria exibir e salvaria todos os pacotes, independente do protocolo. Ao final é solicitado ao usuário um número de pacotes para serem capturados e informado o início da execução do *sniffer*.

```
*******PACKAGE SUMMARY*******
Ether / IP / ICMP 192.168.1.9 > 192.168.1.1 echo-request 0 / Raw

*******PACKAGE DETAILS*******
###[ Ethernet ]###
  dst       = 74:6f:88:f0:70:e7
  src       = 08:00:27:fc:04:c6
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 10899
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x8cbb
     src       = 192.168.1.9
     dst       = 192.168.1.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xdbac
        id        = 0xfc3e
        seq       = 0x1
###[ Raw ]###
           load      = 'bsAe\x00\x00\x00\x00\xb1g\x0c\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

None
*******PACKAGE END*******
```

Figura 24 – Terminal de saída do Sniffer

Fonte: Produção própria

Como pode ser visto, na Figura 24 o pacote capturado será exibido em resumo no início e depois em detalhes. Ao final é informado que o pacote foi encerrado, para facilitar a separação e leitura dos pacotes no terminal.

4.4.6.3 Monitor

Terminal completo

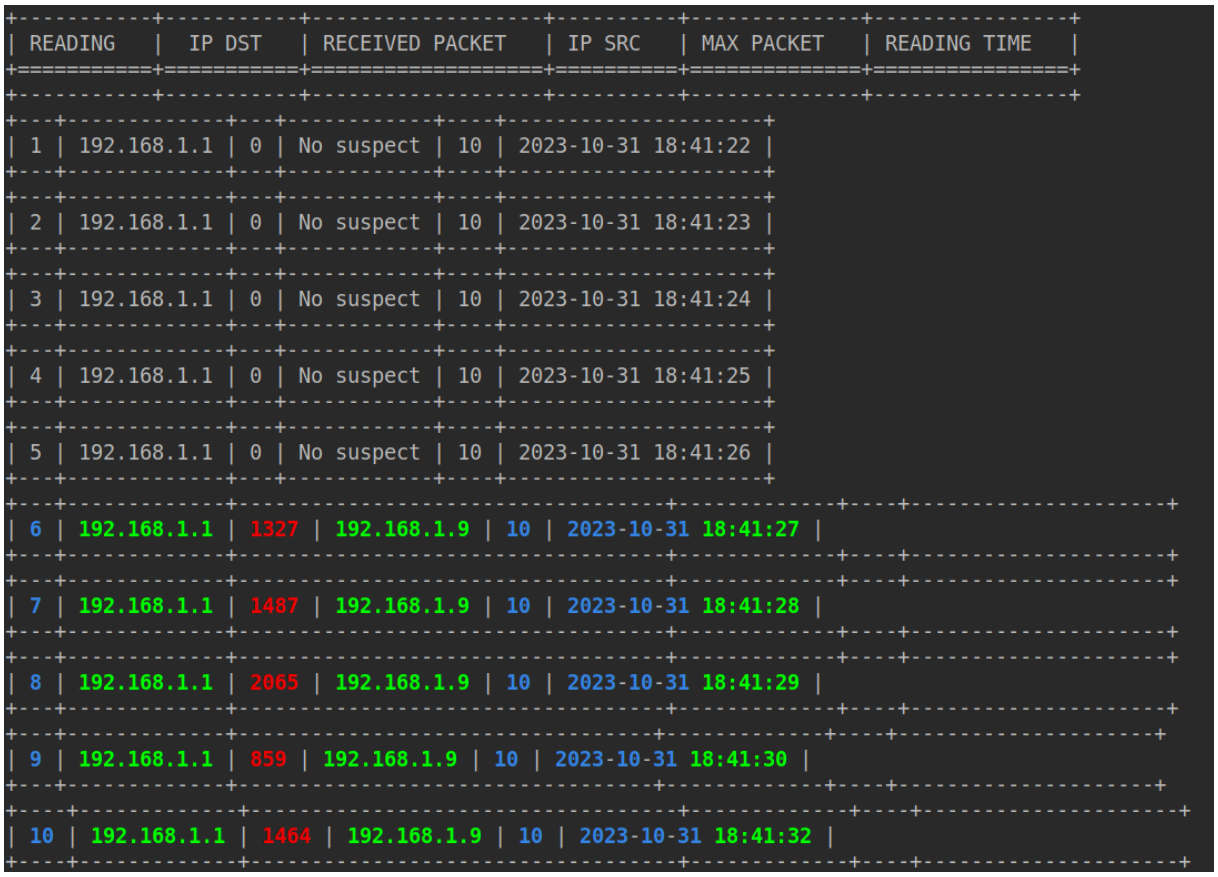


Figura 25 – Terminal de saída do Monitor

Fonte: Produção própria

A Figura 25 mostra como ficará a saída no terminal com a execução do monitor de anomalias de tráfego. Como pode ser visto os dados são separados em colunas de contador, endereço de destino, número de pacotes identificados com destino ao aparelho monitorado, endereço de origem, número máximo de pacotes que são permitidos antes de considerar suspeito, no exemplo o valor é 10, e horário da leitura.

Na Figura 26 pode ser visto a saída em formato de planilha que o código 'Monitor' gera das ocorrências onde o máximo de pacotes para o foi excedido. Contendo a informação na mesma estrutura que a do terminal da figura 25.

No gráfico contido na Figura 27 pode ser visto a leitura feita pelo Monitor executado em conjunto com o lançador no modo de sobrecarga, que é quando o código

Destiny	Source	Number of packages	Seconds	Maximum amount	Reading moment
192.168.1.1	192.168.1.17 204.79.197.203	807	1	50	2023-11-13 09:08:32
192.168.1.1	192.168.1.17 204.79.197.203	836	1	50	2023-11-13 09:08:33
192.168.1.1	192.168.1.17 204.79.197.203	981	1	50	2023-11-13 09:08:34
192.168.1.1	192.168.1.17 204.79.197.203	869	1	50	2023-11-13 09:08:35
192.168.1.1	192.168.1.17 204.79.197.203	889	1	50	2023-11-13 09:08:36
192.168.1.1	192.168.1.17 204.79.197.203	894	1	50	2023-11-13 09:08:37
192.168.1.1	192.168.1.17 204.79.197.203	779	1	50	2023-11-13 09:08:38
192.168.1.1	192.168.1.17 204.79.197.203	968	1	50	2023-11-13 09:08:40
192.168.1.1	192.168.1.17 204.79.197.203	825	1	50	2023-11-13 09:08:41
192.168.1.1	192.168.1.17 204.79.197.203	895	1	50	2023-11-13 09:08:42
192.168.1.1	192.168.1.17 204.79.197.203	826	1	50	2023-11-13 09:08:43
192.168.1.1	192.168.1.17 204.79.197.203	788	1	50	2023-11-13 09:08:44
192.168.1.1	192.168.1.17 204.79.197.203	799	1	50	2023-11-13 09:08:45
192.168.1.1	192.168.1.17 204.79.197.203	917	1	50	2023-11-13 09:08:47
192.168.1.1	192.168.1.17 204.79.197.203	828	1	50	2023-11-13 09:08:48
192.168.1.1	192.168.1.17 204.79.197.203	915	1	50	2023-11-13 09:08:49
192.168.1.1	192.168.1.17 204.79.197.203	763	1	50	2023-11-13 09:08:50

Figura 26 – Saída do monitor de anomalia em planilha

Fonte: Produção própria

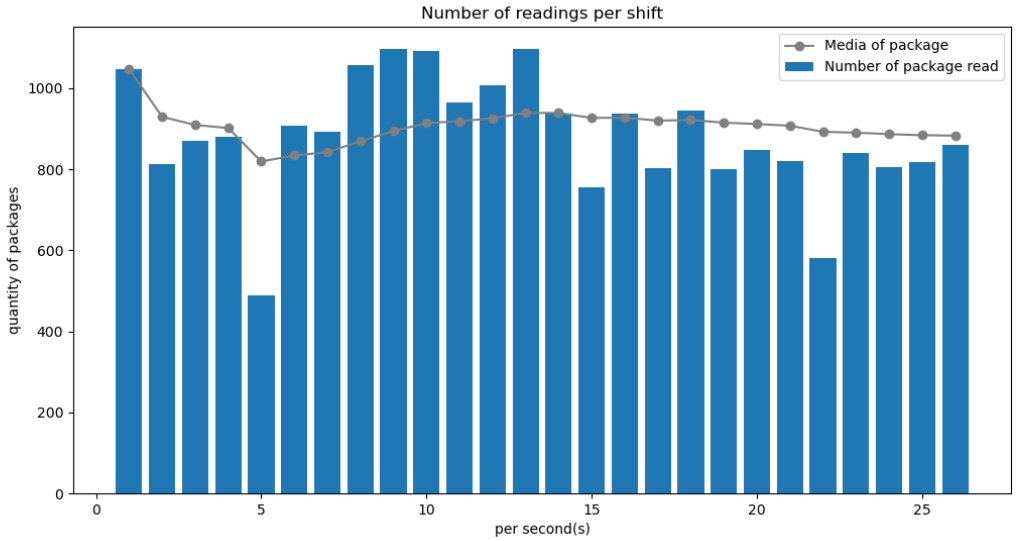


Figura 27 – Leitura do Monitor com Lançador em modo sobrecarga

Fonte: Produção própria

envia o máximo de pacotes que conseguir. Pode ser observado no gráfico o contador da leitura em barras azuis e a média aritmética com uma linha cinza, que fica geralmente entre 800 e 1000 pacotes identificados.

A Figura 28 mostra a quantidade de pacotes lidos pelo Monitor quando executado o hping3 usando o parâmetro –faster, que envia o máximo de pacotes que a ferramenta conseguir. Pode ser visto um pico de 2000 pacotes e um mínimo por volta de 550, com a média ficando acima dos 1000 pacotes. Como pode ser notado a leitura teve uma abrupta queda nas leituras, isso ocorreu porque o hping3 foi encerrado e executado novamente em seguida.

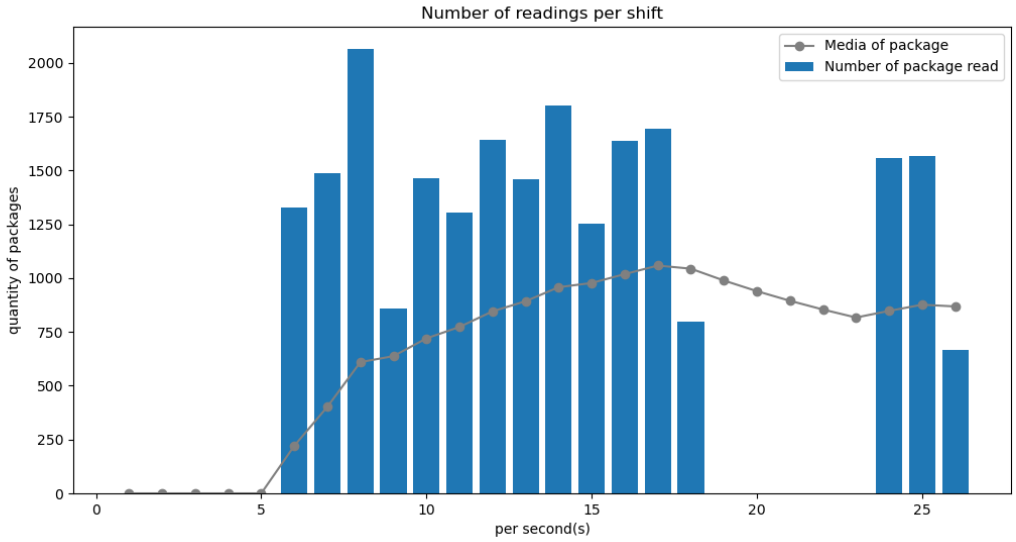


Figura 28 – Leitura do Monitor com hping3

Fonte: Produção própria

4.4.6.4 Lançador

```
Select a supported packet type (TCP, UDP, ICMP): icmp
Enter the IP address of the destination host: 192.168.1.1
1
Enter a payload to be in the package: Mensagem de exemplo
Want to use random MAC and IP address? (Y for yes): N
Select a available interface (lo, enp0s3, docker0): enp0s3
###[ Ethernet ]###
  dst       = 74:6f:88:f0:70:e7
  src       = 08:00:27:fc:04:c6
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = None
     tos       = 0x0
     len       = None
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = None
     src       = 192.168.1.9
     dst       = 192.168.1.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = None
        id        = 0x0
        seq       = 0x0
###[ Raw ]###
           load      = 'Mensagem de exemplo'

None
Ether / IP / ICMP 192.168.1.9 > 192.168.1.1 echo-request 0 / Raw
```

Figura 29 – Terminal de entrada do Lançador

Fonte: Produção própria

A Figura 29 mostra a saída completa em terminal da execução do Lançador. A execução inicia solicitando o protocolo que será usado no lançamento. Logo após isso o código solicita a entrada do endereço IPv4 de destino. O código então pede um texto para ser inserido como payload do pacote. É então perguntado se o usuário deseja que o lançador use um endereço de origem aleatório, ou se quer usar o original da interface. Feito isso o código pede o nome da interface que será utilizada para fazer os envios,

isso conclui as informações utilizadas no pacote, ele é então exibido no terminal. Por último o código pergunta ao usuário qual modo de envio ele deseja utilizar, podendo ser apenas um envio - single, o envio exponencial - exponential, ou se deseja ser enviado em modo de sobrecarga - overload. Os pontos depois disso representam cada pacote sendo enviado.

5 CONCLUSÃO

Este trabalho desenvolveu *Scripts* para *Pentest* redes computacionais que auxiliam o controle de segurança do usuário nas redes, tornando mais prático e simples a visualização de vulnerabilidades que podem prejudicar a segurança dos dados transportados enquanto o usuário está conectado a uma rede.

Foi realizada uma revisão e pesquisa bibliográfica sobre o roubo e sequestro de dados que vem se tornando cada vez mais presente em nossa sociedade, onde todos os dias estamos nos conectando em redes onde não podemos ter total conhecimento da de quem está conectado ao mesmo tempo com a gente, podendo essas pessoas estarem interessadas nas portas de acesso a aplicações usadas pelo *host*, conhecendo um pouco mais os processos que os desenvolvedores de aplicações e técnicos de rede seguem para aplicar uma rede, podemos criar *scripts* que beneficiam os desenvolvedores e analistas de infraestrutura a entender o que se passa na rede.

Com o passar dos anos, os casos de roubo e sequestro de dados se tornou cada vez mais comuns, sendo que a cada ano um novo recorde de invasões é alcançado por conta da popularização da internet e redes sem fio de acesso comum ou público, com essa disponibilidade de dados sobre redes, os *scripts* vem de encontro a sanar alguns problemas, por exemplo: aplicações sendo executadas em portas comuns, que são frequentemente verificadas, fácil acesso aos dados veiculados na rede por meio de "*sniffing*", análises de dispositivos conectados na rede em busca de vulnerabilidades nas aplicações, entre outros.

Finalmente, após diversas versões e aprimoramentos, os *scripts* se mostraram eficientes e capazes de identificar portas vulnerabilidades de acordo com a aplicação que as consome ou problemas na rede de uma maneira eficiente, usual e veloz quando comparado a ferramentas semelhantes encontradas em aplicações de segurança de rede, além de facilitar a compreensão dos dados obtidos durante a execução por conta da exibição, o que permite que pessoas menos técnicas possam utilizar os scripts para avaliar a segurança de seu dispositivo.

5.0.1 Dificuldades encontradas

Ao longo do desenvolvimento do trabalho foram encontradas algumas dificuldades, sendo a primeira delas encontrar qual biblioteca seria melhor para cada código, por essa razão diversas bibliotecas precisaram ser substituídas no meio do desenvolvimento por limitação próprias de cada biblioteca.

Ainda em relação a bibliotecas, outro dificuldade foi de estudar a documentação das mesmas e depois integrar funções de origens diferentes, que quando executas em conjunto causam resultados indesejados, isso ocorreu em grande parte por cada biblioteca ter seu próprio objeto com características especificas.

Outra limitação enfrentada foi na parte do ambiente, onde somente redes de tipo C foram disponibilizadas para os testes, mesmo o código suportando redes de tipo A e B não foi possível fazer teste nelas por falta de permissões das instituições ou empresas que possuem redes dessas dimensões.

Por último uma grande limitação foi depender da autorização de redes reais para os testes, por falta de capacidade computacional capaz de simular uma rede diversificada usando máquinas virtuais.

5.0.2 Trabalhos futuros

Como objetivos para o futuro do trabalho, planejamos fazer a integração dos códigos, tornando possível que o usuário use apenas poucos comandos para a execução dos *scripts*, realizando a leitura de rede, varredura de portas e análise de pacotes rapidamente em uma única interface.

Em conjunto com a integração dos códigos, também é planejado a melhora na exibição dos resultados obtidos, tornando possível que o usuário selecione o melhor método de exibição para o cenário em que se encontra, podendo escolher entre variações de gráficos, formatos de arquivos e podendo adicionar ou remover parâmetros de sua análise.

Adicionar a funcionalidade de escanear redes remotamente, tornando possível que um usuário possa realizar a execução dos *scripts* de maneira remota ou portátil, fazendo com que as possibilidades de uso sejam maiores e mais práticas.

REFERÊNCIAS

CERT.br; NIC.br. **Cartilha de Seguança para Internet.** 2012. Disponível em: <https://cartilha.cert.br/>. Citado na página 12.

Cloudflare . **O que é uma porta de computador? | Portas na rede**. 2022. Disponível em: <https://www.cloudflare.com/pt-br/learning/network-layer/what-is-a-computer-port/#:~:text=As%20portas%20sÃčo%20padronizadas%20em,enviadas%20para%20a%20porta%2080./>. Citado na página 20.

Debian. **Debian**. 2023. Disponível em: <https://www.debian.org/index.pt.html>. Citado na página 29.

IBM . **Protocolo de Resolução de Endereço**. 2023. Disponível em: <https://www.ibm.com/docs/pt-br/aix/7.3?topic=protocols-address-resolution-protocol>. Citado na página 14.

JARGAS, A. M. **Expressões regulares: uma abordagem divertida**. [S.l.]: Editora Novatec, 2012. Citado na página 16.

KUROSE, R. **Redes de Computadores e a Internet: uma abordagem top-down 6ªed.** [S.l.]: São Paulo: Pearson, 2013. Citado 2 vezes nas páginas 14 e 19.

McGugan, W . **Rich's documentation**. 2023. Disponível em: <https://rich.readthedocs.io/en/stable/index.html>. Citado na página 28.

Moody, P . **IpAdress**. 2023. Disponível em: <https://docs.python.org/3/library/ipaddress.htm>. Citado na página 28.

MORENO, D. **Pentest em redes sem fio. 1 ed.** [S.l.]: Novatec, 2016. Citado 2 vezes nas páginas 20 e 23.

Parrot. **Parrot OS**. 2023. Disponível em: <https://www.parrotsec.org/>. Citado na página 29.

PyPi . **python-nmap**. 2021. Disponível em: <https://pypi.org/project/python-nmap/>. Citado 2 vezes nas páginas 21 e 27.

PyPi . **tabulate**. 2022. Disponível em: <https://pypi.org/project/tabulate/>. Citado na página 28.

PyPi . **matplotlib**. 2023. Disponível em: <https://pypi.org/project/matplotlib/>. Citado na página 28.

PyPi . **termcolor**. 2023. Disponível em: <https://pypi.org/project/termcolor/>. Citado na página 29.

Python Documentation . **Diversas interfaces de sistema operacional**. 2022. Disponível em: <https://docs.python.org/pt-br/3.7/library/os.html>. Citado na página 29.

Python Documentation . **DateTime**. 2023. Disponível em: <https://pypi.org/project/DateTime/>. Citado na página 29.

Python Documentation . **Time**. 2023. Disponível em: <https://docs.python.org/3/library/time.html>. Citado na página 29.

Scapy. **What is Scapy?** 2022. Disponível em: <https://scapy.net>. Citado na página 28.

TANENBAUM. **Redes de Computadores. 5ª ed**. [S.l.]: São Paulo: Pearson, 2011. Citado 3 vezes nas páginas 15, 16 e 18.

Tecnoblog. **O que é TCP/IP?** 2022. Disponível em: <https://tecnoblog.net/responde/o-que-e-tcp-ip/>. Citado na página 18.

TP-Link . **Receptor Wi-fi**. 2023. Disponível em: <https://www.tp-link.com/br/home-networking/adapter/tl-wn822n//>. Citado na página 30.

WEIDMAN, G. **Testes de Invasão: Uma introdução prática ao hacking.** [S.l.]: Editora Novatec, 2014. Citado na página 20.