RELATÓRIO – ROTEIRO 1: ESCANEAMENTO DE PORTAS (PORT SCANNER)

Felipe Maia

1. Introdução

Este relatório apresenta o desenvolvimento e a execução de uma ferramenta de escaneamento de portas em Python, implementada com interface gráfica, para o Roteiro 1. A aplicação tem como finalidade identificar portas abertas em um host, relacionar essas portas com serviços conhecidos (well-known ports) e realizar banner grabbing para coletar informações adicionais dos serviços em execução.

---

2. Objetivos do Projeto

O projeto visa implementar uma ferramenta que:

1. Seja desenvolvida em Python.
2. Possua uma interface amigável (GUI) utilizando Tkinter, facilitando o uso pelo usuário.
3. Permita o escaneamento de um host (ou, alternativamente, de uma rede) mediante a inserção do endereço IP e de um intervalo de portas.
4. Mapeie portas well-known – isto é, associe números de portas a serviços conhecidos – e exiba essa relação na saída do programa.
5. Indique o estado das portas (aberta, fechada ou filtrada) durante o escaneamento.
6. Ofereça a opção de escaneamento tanto para TCP quanto para UDP.
7. Realize banner grabbing para capturar e exibir informações enviadas pelos serviços, contribuindo para a identificação do sistema operacional ou versão do serviço.
8. Forneça suporte a IPv6, reconhecendo o formato do endereço e criando o socket adequado.

---

3. Ambiente e Configuração

- **Plataforma de Teste:**
    - Kali Linux em máquina virtual (VirtualBox) configurada em modo Bridge, garantindo que a VM (com IP 172.20.10.6) esteja na mesma faixa de rede que o host alvo.
- **Testes de Conectividade:**
    - Comando `ip a`: Utilizado para listar as interfaces e confirmar que a interface `eth0` possui o IP 172.20.10.6, além dos endereços IPv6.
    - Comando `ping 172.20.10.4`: Confirmou a conectividade entre o Kali Linux e o host alvo (tempo médio ~0.94 ms).
    - Comando `ping 192.168.56.1`: Demonstrou que este endereço não estava acessível (100% de perda de pacotes).

---

4. Metodologia e Desenvolvimento

4.1 Referência com o Nmap

Antes do desenvolvimento, foram utilizados comandos do Nmap para obter uma referência dos resultados esperados:

- `nmap -sT 172.20.10.4`: Realizou o escaneamento TCP e identificou portas abertas como 135, 139, 445, 1309, 3306, 5432 e 8080.
- `nmap -O 172.20.10.4`: Tentou identificar o sistema operacional, apontando uma provável configuração Windows.
- `nmap -sV 172.20.10.4`: Identificou serviços e suas versões (ex.: MySQL 8.0.39 na porta 3306, Apache HTTPd na porta 8080).

4.2 Desenvolvimento do Código em Python

1. *Bibliotecas Utilizadas:*
    - **socket:** Para estabelecer conexões TCP e UDP e realizar banner grabbing.
    - **tkinter:** Para criar a interface gráfica (GUI) com elementos de entrada, botões e área de resultados.

2. *Interface Gráfica (GUI):*
    - A **GUI** permite ao usuário inserir o endereço IP do alvo e o intervalo de portas a ser escaneado (por exemplo, “20-80”).
    - É possível selecionar o protocolo (TCP ou UDP) por meio de botões de opção (radio buttons).
    - Um botão “Iniciar Escaneamento” aciona a varredura, e os resultados são exibidos em uma área de texto com barra de rolagem.
3. *Mapeamento das Well-Known Ports:*
    - Foi criado um dicionário que mapeia números de portas a nomes de serviços conhecidos.
    - **Exemplo:**

```
WELL_KNOWN_PORTS = {
    22: "SSH",
    80: "HTTP",
    443: "HTTPS",
    3306: "MySQL",
    5432: "PostgreSQL",
    135: "MSRPC",
    139: "NetBIOS-SSN",
    445: "Microsoft-DS",
    8080: "HTTP-Proxy",
    1309: "tcpwrapped"
}
```

    - Durante o escaneamento, o programa verifica se a porta escaneada está presente nesse dicionário e exibe o nome do serviço associado. Caso contrário, exibe “Serviço não identificado”. Essa abordagem permite ao usuário identificar rapidamente os serviços associados às portas abertas.
4. *Escaneamento TCP e UDP:*
    - **TCP:** Para cada porta, o script tenta estabelecer uma conexão com `socket.connect_ex`. Se a conexão for bem-sucedida (retorno 0), a porta é

considerada aberta e o script tenta realizar banner grabbing. Se não, é marcada como “fechada ou filtrada”.

- **UDP:** Envia um pacote vazio e aguarda uma resposta. Se não houver resposta em um tempo definido (timeout), a porta é considerada fechada ou filtrada; se houver resposta, é marcada como aberta.

5. *Banner Grabbing:*
   - Se a porta estiver aberta, o script tenta capturar até 1024 bytes enviados pelo serviço logo após a conexão. Essa informação (banner) é exibida ao usuário, permitindo a identificação de versões ou pistas sobre o sistema operacional.
6. *Suporte a IPv6:*
   - O código detecta se o endereço IP contém “:” e, assim, utiliza o socket AF_INET6 para endereços IPv6, garantindo compatibilidade com ambos os formatos.

---

5. Execução e Resultados

5.1 Procedimento de Execução

- **Passos para executar o portscanner.py:**
  1. *Abra o terminal e navegue até o diretório onde o arquivo está localizado.*
  2. *Execute o script com:*

     ```
     python portscanner.py
     ```

  3. *Na interface que se abre, insira:*
     - Endereço IP: (exemplo: 172.20.10.4)
     - Intervalo de Portas: (exemplo: 20-80)
     - Protocolo: Selecione entre TCP ou UDP.
  4. *Clique em “Iniciar Escaneamento” e aguarde os resultados que serão exibidos na área de texto.*

5.2 Resultados Observados

- Estado das Portas:
  As portas no intervalo informado são verificadas e marcadas como “abertas” se a conexão foi estabelecida, ou “fechadas ou filtradas” caso contrário.
- Mapeamento de Serviços:
  Para portas mapeadas no dicionário, o programa exibe o número da porta seguido do serviço associado (ex.: porta 80 → HTTP, porta 22 → SSH).
- Banner Grabbing:
  Quando aplicável, o banner é exibido logo abaixo do status da porta, permitindo que o usuário visualize informações sobre o serviço.
- Comparação com Nmap:
  Os resultados obtidos pelo port scanner foram comparados aos resultados do Nmap (que detectou portas como 135, 139, 445, 3306, 5432 e 8080). A ferramenta em Python mostrou coerência com esses resultados, validando sua funcionalidade.

---

6. Atendimento aos Requisitos

Para garantir a nota máxima, o projeto foi desenvolvido considerando os seguintes requisitos, com as respectivas funcionalidades implementadas:

1. *Linguagem Python:*
   - O código está totalmente implementado em Python.
2. *Interface Amigável (1 ponto):*
   - A GUI foi construída com Tkinter, permitindo fácil inserção de dados e visualização dos resultados.
3. *Escaneamento de Host ou Rede (1 ponto):*
   - O usuário insere o IP do alvo. Embora o código faça escaneamento de um único host, a funcionalidade permite escanear qualquer host na rede.
4. *Intervalo de Portas (1 ponto):*
   - O usuário define o intervalo de portas a ser escaneado (ex.: “20-80”).

5. *Mapeamento de Well-Known Ports (2 pontos):*
    - Um dicionário mapeia portas conhecidas aos seus respectivos serviços, e essa informação é exibida na saída.
6. *Detecção do Estado das Portas (1 ponto):*
    - O programa informa se a porta está aberta, fechada ou filtrada, baseado na resposta do socket.
7. *Opção de Escaneamento UDP (2 pontos):*
    - Implementação da função scan_udp e a opção para o usuário selecionar UDP na interface.
8. *Banner Grabbing (2 pontos):*
    - O código realiza a captura do banner em portas abertas, exibindo-o na interface, o que pode ser utilizado para dedução do sistema operacional ou versão do serviço.
9. *Suporte a IPv6 (2 pontos):*
    - A função get_socket_family identifica se o endereço é IPv6 e cria o socket apropriado.

---

7. Conclusão

O port scanner desenvolvido cumpre os objetivos do Roteiro 1 ao oferecer uma ferramenta robusta para a varredura de portas em hosts específicos. Com funcionalidades que incluem interface gráfica, mapeamento de well-known ports, escaneamento tanto para TCP quanto para UDP, banner grabbing e suporte a IPv6, a aplicação não só valida os conceitos teóricos estudados, mas também fornece uma base prática para futuras expansões – como a varredura de redes inteiras ou a implementação de análise mais sofisticada de banners para identificação de sistemas operacionais.