Universidade de Pernambuco - UPE
Escola Politécnica de Pernambuco - POLI
Engenharia de computação
Redes de computadores 2
Prof. Edison de Queiroz Albuquerque

Sniffer - tpcdumPoli

Erick Veríssimo Andrade da Silva
Lucas Azevêdo da Silva
Lucas Andreoni
Rodrigo Luiz da Silva

RECIFE - PE
2022

Motivação

Ao se pensar na utilidade do sniffer (Farejador), no universo de redes de computadores, várias aplicações podem ser pontuadas, desde a utilização do sniffer para manter a estabilidade do tráfego em um rede, com a verificação do tráfego, ou a utilização para captura e análise de pacotes por um Black Hat (Chapéu preto), violando assim os princípios de segurança da informação. Desse modo, tanto para o bem como para o mau, é evidente a função do sniffer, monitorar o tráfego de internet em tempo real, capturando todos os dados que entram e saem de um computador.
Assim, com o objetivo de entender a funcionalidade do sniffer e como o modelo OSI atua nessa aplicação, desenvolvemos um sniffer, na linguagem de programação python, para a disciplina de redes de computadores 2, ministrada pelo professor Edison de Queiroz Albuquerque.

Ferramentas utilizadas

- Linguagem: Python 3.10
- Bibliotecas:
    - Socket
    - Struct
    - Textwrap

Como funciona

Inicialmente o código irá capturar todos os pacotes que transitam pela placa de rede, ao detectar um pacote, o sniffer realizará a análise do mesmo.

Um detalhe importante sobre o programa é que ele está em modo promíscuo, logo ele irá receber todos os pacotes que passarem pela rede.

Como executar

python3 sniffer.py ou sudo python3 sniffer.py (para caso o terminal peça permissão)

Código

```
import socket
import struct
import textwrap

# Apenas variáveis para ajudar na formatação de texto
TAB_1 = '\t - '
TAB_2 = '\t\t - '
TAB_3 = '\t\t\t - '
TAB_4 = '\t\t\t\t - '

DATA_TAB_1 = '\t - '
DATA_TAB_2 = '\t\t - '
DATA_TAB_3 = '\t\t\t - '
DATA_TAB_4 = '\t\t\t\t - '

def main():
    # Abertura do socket
    # ntohs = garante que ele seja compatível com qualquer máquina
            conn   =   socket.socket(socket.AF_PACKET,   socket.SOCK_RAW,
socket.ntohs(3))

    while True:
         # Escuta qualquer informação que passar, (conteúdo (raw_data) e
endereço (addr))
        raw_data, addr = conn.recvfrom(65536)
        # Passamos a informação para ser desempacotada
        dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)

print('\n-----------------------------------------------------------------
---------------------')
        print('\nEthernet Frame:')
           # Apenas um print formatado com as informações recebidas do
tráfego
                print(TAB_1  +  'Destino:  {},  Origem:  {},  Protocolo:
{}'.format(dest_mac, src_mac, eth_proto))

            #Antes de printar o conteúdo, devemos checar o protocolo
ethernet (8 para IPv4)
        if eth_proto == 8:
```

```
            (version, header_length, ttl, proto, src, target, data) =
unpack_ipv4(data)
            print(TAB_1 + 'Pacote IPv4:')
              print(TAB_2 + 'Versão: {}, Tamanho do cabeçalho: {}, TTL:
{}'.format(version, header_length, ttl))
                  print(TAB_2 + 'Protocolo: {}, Origem: {}, Target:
{}'.format(proto, src, target))

            # Checagem do protocolo do conteúdo:
            # 1 - para ICMP
            # 6 - para TCP
            # 17 - para UDP
            if proto == 1:
                icmp_type, code, checksum, data = unpack_icmp(data)
                print(TAB_1 + 'Pacote ICMP:')
                       print(TAB_2 + 'Tipo: {}, Código: {}, Checksum:
{}'.format(icmp_type, code, checksum))
                print(TAB_2 + 'Data:')
                print(format_multi_line(DATA_TAB_3, data))

print('\n------------------------------------------------------------
----------------------')

            elif proto == 6:
                       src_port, dest_port, sequence, acknowledgement,
flag_urg, flag_ack, flag_psh, flag_rst, flag_syn, flag_fin, data =
unpack_tcp(data)
                print(TAB_1 + 'Pacote TCP:')
                  print(TAB_2 + 'Porta de origem: {}, Porta de destino:
{}'.format(src_port, dest_port))
                        print(TAB_2 + 'Sequência: {}, Reconhecimento:
{}'.format(sequence, acknowledgement))
                print(TAB_2 + 'Flags:')
                             print(TAB_3 + 'URG: {}, ACK: {}, PSH:
{}'.format(flag_urg, flag_ack, flag_psh))
                             print(TAB_3 + 'RST: {}, SYN: {}, FIN:
{}'.format(flag_rst, flag_syn, flag_fin))
                print(TAB_2 + 'Data:')
                print(format_multi_line(DATA_TAB_3, data))

print('\n------------------------------------------------------------
----------------------')
```

```
            elif proto == 17:
                src_port, dest_port, size, data = unpack_udp(data)
                print(TAB_1 + 'Pacote UDP:')
                  print(TAB_2 + 'Porta de origem: {}, Porta de destino:
{}, Tamanho: {}'.format(src_port, dest_port, size))

print('\n----------------------------------------------------------------
---------------------')

            else:
                print(TAB_1 + 'Data:')
                print(format_multi_line(DATA_TAB_2, data))

print('\n----------------------------------------------------------------
---------------------')

        else:
            print('Data:')
            print(format_multi_line(DATA_TAB_1, data))

print('\n----------------------------------------------------------------
---------------------')

# Desempacota o dado passado
def ethernet_frame(data):
    # desempacota para o formato desejado
      # Basicamente iremos ler 14bytes de pacote, 6 de entrada, 6 de
saída e 2 para o fim;
    dest_mac, src_mac, proto = struct.unpack('! 6s 6s H', data[:14])
    # dest_mac -> para onde vai 6s = 6 caracteres ou 6 bytes
    # src_mac -> de onde vem 6s = 6 caracteres ou 6 bytes
      # proto (tipo ethernet ou protocolo ethernet) H => unsigned int
(int positivo)
    # data[14:] é o conteúdo em si
              return    get_mac_addr(dest_mac),    get_mac_addr(src_mac),
socket.htons(proto), data[14:]

# Retorna de forma apropriada o endereço
def get_mac_addr(bytes_addr):
    # Iremos iterar cada pedaço do endereço e separa-los em pedaços
    bytes_str = map('{:02x}'.format, bytes_addr)
```

```
    # Aqui iremos formatar o endereço para que ele retorne no formato correto, ex: '00:00:00:00:00:00'
    return ':'.join(bytes_str).upper()


# Desempacotando cabeçalhos de pacotes IPv4
def unpack_ipv4(data):
    # Primeiro iremos desempacotar a versão e o tamanho do header
    version_header_length = data[0]
    # pegando apenas o número da versão:
    version = version_header_length >> 4
    # pegando o tamanho do header (precisamos multiplicar por 4, pois, iremos comparar dois bytes e pegá-los quando ambos forem 1):
    header_length = (version_header_length & 15) * 4
    # Agora podemos começar desempacotar tudo, é necessário pegar o header_length antes pois é isso que determina quando o conteúdo começa
    # Logo após o fim do header, tudo será apenas "data" (conteúdo)
    # tamanho  que será de 20bytes e formatação do conteúdo ('! 8x B B 2x 4s 4s')
    ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
    return version, header_length, ttl, proto, ipv4(src), ipv4(target), data[header_length:]

# Retorna em formato correto o endereço IPv4 (Ex: 127.0.0.1)
def ipv4(addr):
    return '.'.join(map(str, addr))


# Abaixo ficam as funções que desempacotam os pacotes de acordo com seu tipo (UDP, TCP, ICMP)

# Desempacota ICMP:
def unpack_icmp(data):
    icmp_type, code, checksum = struct.unpack('! B B H', data[:4])
    return icmp_type, code, checksum, data[4:]

# Desempacota TCP:
def unpack_tcp(data):
    # Aqui serão também puxadas todas as 'flags' que o protocolor TCP tem
```

```
                (src_port,    dest_port,    sequence,    acknowledgement,
offset_reserved_flags) = struct.unpack('! H H L L H', data[:14])
    offset = (offset_reserved_flags >> 12) * 4
    flag_urg = (offset_reserved_flags & 32) >> 5
    flag_ack = (offset_reserved_flags & 16) >> 4
    flag_psh = (offset_reserved_flags & 8) >> 3
    flag_rst = (offset_reserved_flags & 4) >> 2
    flag_syn = (offset_reserved_flags & 2) >> 1
    flag_fin = offset_reserved_flags & 1
      return src_port, dest_port, sequence, acknowledgement, flag_urg,
flag_ack, flag_psh, flag_rst, flag_syn, flag_fin, data[offset:]


# Desempacota UDP:
def unpack_udp(data):
    src_port, dest_port, size = struct.unpack('! H H 2x H', data[:8])
    return src_port, dest_port, size, data[8:]

# Formata uma string que seja muito longa, transformando ela em linha
por linha:
def format_multi_line(prefix, string, size=80):
    size -= len(prefix)
    if isinstance(string, bytes):
        string = ''.join(r'\x{:02x}'.format(byte) for byte in string)
        if size % 2:
            size -= 1

     return '\n'.join([prefix + line for line in textwrap.wrap(string,
size)])

main()
```