



# Analizando Tráfego de Redes e *Sniffing* de Pacotes

## Parte II

Prof. Ricardo Mesquita

# Consulta DNS

- Podemos realizar consultas DNS para obter servidores de nomes de domínio.
- No exemplo a seguir construímos um pacote com as camadas IP, UDP e DNS com o nome de domínio a ser consultado, então, enviamos este pacote e obtemos o pacote de resposta.

```
from scapy.all import *
def queryDNS(dnsServer, dominio):
    packet_dns=IP(dst=dnsServer)/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname=dominio))
    response_packet = sr1(packet_dns, verbose=1)
    print(response_packet.show()) return response_packet[DNS].summary()
if __name__ == "__main__":
    print (queryDNS("8.8.8.8", "www.python.org"))
```

Observe a estrutura do pacote de consulta DNS, que é um pacote UDP na porta 53, e o servidor de nomes e domínio fornecidos.

Executando o script anterior, podemos ver o servidor de nomes do domínio [www.python.org](http://www.python.org).

# Consulta DNS

```
$ sudo python scapy_query_dns.py
Begin emission:
Finished sending 1 packets.
Received 2 packets, got 1 answers, remaining 0 packets
###[ IP ]###
  version = 4
  ihl = 5
  tos = 0x0
  len = 121
  id = 57690
  flags =
  frag = 0
  ttl = 122
  proto = udp
  checksum = 0x7bd2
  src = 8.8.8.8
  dst = 192.168.18.143
  \options \
```

```
###[ UDP ]###
  sport = domain
  dport = domain
  len = 101
  checksum = 0xbde9
```

# Consulta DNS

```
###[ DNS ]###
```

```
id          = 0
qr          = 1
opcode      = QUERY
aa          = 0
tc          = 0
rd          = 1
ra          = 1
z           = 0
ad          = 0
cd          = 0
rcode       = ok
qdcount     = 1
ancount     = 2
nscount     = 0
arcount     = 0
\qd \
```

None

DNS Ans "b'dualstack.python.map.fastly.net.'"

```
###[ DNS Question Record ]###
| qname = 'www.python.org.'
| qtype = A
| qclass = IN
\an \
###[ DNS Resource Record ]###
| rname = 'www.python.org.'
| type = CNAME
| rclass = IN
| ttl = 21572
| rdlen = None
| rdata = 'dualstack.python.map.fastly.net.' |
###[ DNS Resource Record ]###
| rname = 'dualstack.python.map.fastly.net.'
| type = A
| rclass = IN
| ttl = 2
| rdlen = None
| rdata = 151.101.132.223
ns = None
ar = None
```

# Descoberta de Rede com scapy

- Podemos criar um pacote ICMP na camada IP e enviá-lo pela rede usando o método `sr1()`:

```
>>> test_icmp = sr1(IP(dst="45.33.32.156")/ICMP())
```

```
Begin emission:
```

```
Finished sending 1 packets.
```

```
.*
```

```
Received 2 packets, got 1 answers, remaining 0 packets
```

# Descoberta de Rede com scapy

- Podemos ver os resultados da resposta usando o método **display()** e a variável **test\_icmp**:

```
>>> test_icmp.display()
###[ IP ]###
  version = 4
  ihl = 5
  tos = 0x28
  len = 28
  id = 62692
  flags =
  frag = 0
  ttl = 44
  proto = icmp
  checksum = 0x795a
  src = 45.33.32.156
  dst = 192.168.18.21
  \options \
```

```
###[ ICMP ]###
  type = echo-reply
  code = 0
  checksum = 0x0
  id = 0x0
  seq = 0x0
  unused = ''
```

# Descoberta de Rede com scapy

- Com o script a seguir, podemos verificar se um host está ativo ou não.

```
import sys
from scapy.all import *
target = sys.argv[1]
icmp = IP(dst=target)/ICMP()
recv = sr1(icmp, timeout=10)
if recv is not None:
    print("Target IP is live")
```

# Descoberta de Rede com scapy

- Ao executar o script, vemos, na saída, informações sobre os pacotes recebidos.

```
$ sudo python scapy_icmp_target.py 45.33.32.156
```

```
Begin emission:
```

```
Finished sending 1 packets.
```

```
.....*
```

```
Received 60 packets, got 1 answers, remaining 0 packets
```

```
Target IP is live
```



# Descoberta de Rede com scapy

- Outro método que podemos usar para verificar hosts ativos para redes internas e externas é o método TCP SYN ping.

```
from scapy.all import *  
target = sys.argv[1]  
port = int(sys.argv[2])  
ans,unans = sr(IP(dst=target)/TCP(dport=port,flags="S"))  
ans.summary()
```

# Descoberta de Rede com scapy

- No script anterior, usamos o método sr() para enviar o pacote e receber uma resposta:

```
$ sudo python scapy_tcp_target.py 45.33.32.156 80
```

```
Begin emission:
```

```
Finished sending 1 packets.
```

```
.....*
```

```
Received 16 packets, got 1 answers, remaining 0 packets
```

```
IP / TCP 192.168.18.21:ftp_data > 45.33.32.156:www_http S ==> IP / TCP  
45.33.32.156:www_http > 192.168.18.21:ftp_data SA
```

# Escaneamento de Porta com scapy

- No exemplo a seguir, vamos definir o método **analyze\_port()**, que fornece os parâmetros *host*, *port* e *verbose\_level*.
- Este método será responsável por enviar um pacote TCP e aguardar sua resposta.
- Ao processar a resposta, o objetivo será o de verificar, dentro da camada TCP, se o flag recebido corresponde a uma porta em estado aberto, fechado ou filtrado.

```
mirror_mod = modifier_ob.  
Part mirror object to mirror  
mirror_mod.mirror_object  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
  
selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
  
print("please select exactly  
  
-- OPERATOR CLASSES ----  
  
types.Operator):  
on X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
  
context):  
context.active_object is not
```

# Escaneamento de Porta com scapy

```
import sys
from scapy.all import *import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
def analyze_port(host, port, verbose_level):
    print("[+] Scanning port %s" % port)
    packet = IP(dst=host)/TCP(dport=port,flags="S")
    response = sr1(packet,timeout=0.5,verbose=verbose_level)
    if response is not None and response.haslayer(TCP):
        if response[TCP].flags == 18:
            print("Port "+str(port)+" is open!")
            sr(IP(dst=target)/TCP(dport=response.sport,flags="R"),timeout=0.5, verbose=0)
        elif response.haslayer(TCP) and response.getlayer(TCP).flags == 0x14:
            print("Port:"+str(port)+" Closed")
        elif response.haslayer(ICMP):
            if(int(response.getlayer(ICMP).type)==3 and int(response.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                print("Port:"+str(port)+" Filtered")
```

# Escaneamento de Porta com scapy

- Em nosso programa principal, gerenciamos os parâmetros relacionados ao host e intervalo de portas e outro parâmetro que indica o nível de depuração:

```
if __name__ == '__main__':  
    if len(sys.argv) != 5:  
        print("usage: %s target startport endpoint verbose_level" % (sys.argv[0]))  
        sys.exit(0)  
    target = str(sys.argv[1])  
    start_port = int(sys.argv[2])  
    end_port = int(sys.argv[3])+1  
    verbose_level = int(str(sys.argv[4]))  
    print("Scanning "+target+" for open TCP ports\n")  
    for port in range(start_port, end_port):  
        analyze_port(target, port, verbose_level)
```

# Escaneamento de Porta com scapy

- Ao executar o script anterior sobre um host específico e em um intervalo de portas, ele verifica o status de cada porta e exibe o resultado na tela:

```
$ sudo python scapy_port_scan.py scanme.nmap.org 20 23 0
Scanning scanme.nmap.org for open TCP ports
[+] Scanning port 20
Port:20 Closed
[+] Scanning port 21
Port:21 Closed
[+] Scanning port 22
Port 22 is open!
[+] Scanning port 23
Port:23 Closed
Scan complete!
```

# Escaneamento de Porta com scapy

- Também temos a opção de executar o script e mostrar um maior nível de detalhe se usarmos o último parâmetro `verbose_level=1`.

```
$ sudo python scapy_port_scan.py scanme.nmap.org 79 80 1
Scanning scanme.nmap.org for open TCP ports
[+] Scanning port 79
Begin emission:
Finished sending 1 packets.
    Received 20 packets, got 1 answers, remaining 0 packets
Port:79 Closed
[+] Scanning port 80
Begin emission:
Finished sending 1 packets.
    Received 10 packets, got 1 answers, remaining 0 packets
Port 80 is open!
Scan complete!
```



# Traceroute com scapy

- Cada pacote transmitido possui um atributo TTL.
  - Isso ajuda a listar os roteadores pelos quais o pacote passa para chegar à máquina de destino.
  - Quando uma máquina recebe um pacote IP, ela diminui o atributo TTL em 1 e depois o repassa.
- Se o TTL do pacote acabar antes de ele responder, a máquina alvo enviará um pacote ICMP com uma mensagem de falha.
- O scapy fornece uma função integrada para tracerouting conforme mostrado no exemplo a seguir.





```
>>> traceroute("45.33.32.156")
```

```
Begin emission:
```

```
Finished sending 30 packets.
```

```
*****
```

```
Received 28 packets, got 28 answers, remaining 2 packets
```

```
45.33.32.156:tcp80
```

```
1  192.168.18.1          11
```

```
3  192.168.210.40       11
```

```
4  192.168.209.117     11
```

```
6  154.54.61.129       11
```

```
7  154.54.85.241       11
```

```
8  154.54.82.249       11
```

```
9  154.54.6.221        11
```

```
10 154.54.42.165       11
```

```
11 154.54.5.89         11
```

```
12 154.54.41.145       11
```

```
13 154.54.44.137       11
```

```
...
```

```
(<Traceroute: TCP:13 UDP:0 ICMP:15 Other:0>,
```

```
<Unanswered: TCP:2 UDP:0 ICMP:0 Other:0>)
```

# Traceroute com scapy

- Ferramentas como o traceroute enviam pacotes com um determinado valor TTL e aguardam a resposta antes de enviar o próximo pacote, o que pode retardar todo o processo, principalmente quando há um nó da rede que não responde.
- Para simular o comando traceroute, poderíamos enviar pacotes ICMP e definir o TTL para 30, o que pode atingir qualquer nó da Internet.
- O valor TTL determina o tempo ou número de saltos que um pacote de dados fará antes que um roteador o rejeite.
- Quando você atribui um TTL ao seu pacote de dados, ele carrega esse número como um valor numérico em segundos.
- Cada vez que o pacote chega a um roteador, o roteador subtrai 1 do valor TTL e o passa para o próximo dispositivo.

# Traceroute com scapy

```
>>> ans,unans = sr(IP(dst="45.33.32.156",ttl=(1,30))/ICMP())
>>> ans.summary(lambda sr:sr[1].sprintf("%IP.src%"))
192.168.18.1
192.168.210.40
10.10.50.51
192.168.209.117
154.54.61.129
154.54.85.241
154.54.82.249
154.54.6.221
154.54.42.165
154.54.5.89
154.54.41.145
154.54.43.70
38.142.11.154
173.230.159.81
154.54.44.137
154.54.1.162
45.33.32.156
```

# Traceroute com scapy

- Para usar scapy com IP e UDP:

```
>>> from scapy.all import *  
>>> ip_packet = IP(dst="google.com", ttl=10)  
>>> udp_packet = UDP(dport=40000)  
>>> full_packet = IP(dst="google.com", ttl=10) / UDP(dport=40000)
```

- Para enviar o pacote:

```
>>> send(full_packet)
```

# Traceroute com scapy

- Para implementar o traceroute, enviamos um pacote UDP com TTL = i para i = 1,2,3, n e verificamos o pacote de resposta para ver se alcançamos o destino e precisamos continuar fazendo saltos para cada host que alcançamos.

```
from scapy.all import *
host = "45.33.32.156"
for i in range(1, 20):
    packet = IP(dst=host, ttl=i) / UDP(dport=33434)
    reply = sr1(packet, verbose=0, timeout=1)
    if reply is None:
        pass
    elif reply.type == 3:
        print("Done!", reply.src)
        break
    else:
        print("%d hops away: " % i , reply.src)
```

# Traceroute com scapy

- Na saída a seguir, podemos ver o resultado da execução do script traceroute.
- O alvo é o endereço IP 45.33.32.156 e podemos ver os saltos até o atingirmos.

```
$ sudo python scapy_traceroute.py
1 hops away: 192.168.18.1
2 hops away: 10.10.50.51
3 hops away: 192.168.210.40
4 hops away: 192.168.209.117
6 hops away: 154.54.61.129
7 hops away: 154.54.85.241
8 hops away: 154.54.82.249
9 hops away: 154.54.6.221
10 hops away: 154.54.42.165
11 hops away: 154.54.5.89
12 hops away: 154.54.41.145
...
17 hops away: 173.230.159.65
Done! 45.33.32.156
```

# Lendo Arquivos pcap com scapy

- PCAP (Packet CAPture): refere-se à API que permite capturar pacotes de rede para processamento.
- O formato PCAP é padrão e é usado por ferramentas de análise de rede conhecidas, como TCPDump, WinDump, Wireshark, TShark e Ettercap.
- O scapy incorpora duas funções para trabalhar com arquivos PCAP:
  - **rdcap()**: Lê e carrega um arquivo .pcap.
  - **wdcap()**: Grava o conteúdo de uma lista de pacotes em um arquivo .pcap.

# Lendo Arquivos pcap com scapy

```
>>> packets = rdpcap('packets.pcap')
```

```
>>> packets.summary()
```

```
Ether / IP / TCP 10.0.2.15:personal_agent > 10.0.2.2:9170 A / Padding
```

```
Ether / IP / TCP 10.0.2.15:personal_agent > 10.0.2.2:9170 PA / Raw
```

```
Ether / IP / TCP 10.0.2.2:9170 > 10.0.2.15:personal_agent A
```

```
Ether / IP / TCP 10.0.2.2:9170 > 10.0.2.15:personal_agent PA / Raw
```

```
Ether / IP / TCP 10.0.2.15:personal_agent > 10.0.2.2:9170 A / Padding
```

```
....
```



# Lendo Arquivos pcap com scapy

```
>>> packets.sessions()
{'ARP 10.0.2.2 > 10.0.2.15': <PacketList: TCP:0 UDP:0 ICMP:0 Other:2>,
 'IPv6 :: > ff02::16 nh=Hop-by-Hop Option Header': <PacketList: TCP:0 UDP:0 ICMP:0
 Other:1>,
 'IPv6 :: > ff02::1:ff12:3456 nh=ICMPv6': <PacketList: TCP:0 UDP:0 ICMP:0 Other:1>,
 'IPv6 fe80::5054:ff:fe12:3456 > ff02::2 nh=ICMPv6': <PacketList: TCP:0 UDP:0 ICMP:0
 Other:3>,
 'ARP 10.0.2.15 > 10.0.2.2': <PacketList: TCP:0 UDP:0 ICMP:0 Other:1>,
 'IPv6 fe80::5054:ff:fe12:3456 > ff02::16 nh=Hop-by-Hop Option Header': <PacketList:
 TCP:0 UDP:0 ICMP:0 Other:1>,
 'TCP 10.0.2.2:9170 > 10.0.2.15:5555': <PacketList: TCP:3338 UDP:0 ICMP:0 Other:0>,
 'TCP 10.0.2.15:5555 > 10.0.2.2:9170': <PacketList: TCP:2876 UDP:0 ICMP:0 Other:0>,
 ... .
```

# Lendo Arquivos pcap com scapy

```
>>> packets.show()
```

```
17754 Ether / IP / TCP 10.0.2.15:personal_agent > 10.0.2.2:9170 A / Padding
```

```
17755 Ether / IP / TCP 10.0.2.15:personal_agent > 10.0.2.2:9170 PA / Raw
```

```
17756 Ether / IP / TCP 10.0.2.2:9170 > 10.0.2.15:personal_agent A
```

```
17757 Ether / IP / TCP 10.0.2.2:9170 > 10.0.2.15:personal_agent PA / Raw
```

```
17758 Ether / IP / TCP 10.0.2.15:personal_agent > 10.0.2.2:9170 A / Padding
```

# Lendo Arquivos pcap com scapy

```
>>> for packet in packets:
...     packet.show()
###[ Ethernet ]###
  dst = ff:ff:ff:ff:ff:ff
  src = cc:00:0a:c4:00:00
  type = IPv4
###[ IP ]###
  version = 4
  ihl = 5
  tos = 0x0
  len = 604
  id = 5
  flags =
  frag = 0
  ttl = 255
  proto = udp
  chksum = 0xb98c
  src = 0.0.0.0
  dst = 255.255.255.255
```

- Para ver em detalhes os dados de um pacote, podemos iterar na lista de pacotes.

# Lendo Arquivos pcap com scapy

```
>>> len(packets)
12
>>> print(packets[0].show())
###[ Ethernet ]###
  dst = ff:ff:ff:ff:ff:ff
  src = cc:00:0a:c4:00:00
  type = IPv4
###[ IP ]###
  version = 4
  ihl = 5
  tos = 0x0
  len = 604
  id = 5
  flags =
  frag = 0
  ttl = 255
  proto = udp
  chksum = 0xb98c
  src = 0.0.0.0
  dst = 255.255.255.255
```

- Também é possível acessar o pacote como se fosse um array ou uma lista:

# Lendo Arquivos pcap com scapy

- O método `get_packet_layer(packet)` nos permite obter as camadas de um pacote:

```
>>> def get_packet_layer(packet):  
...     yield packet.name  
...     while packet.payload:  
...         packet = packet.payload  
...         yield packet.name  
>>> for packet in packets:  
...     layers = list(get_packet_layer(packet))  
...     print("/".join(layers))  
...  
Ethernet/IP/UDP/BOOTP/DHCP options  
.....
```

# Lendo Solicitações DHCP

- Muitos roteadores usam o protocolo DHCP (Dynamic Host Configuration Protocol) para atribuir automaticamente endereços IP a dispositivos de rede.
- No DHCP, o cliente DHCP (dispositivo de rede) primeiro envia uma mensagem de descoberta de DHCP para todos os destinos na LAN para consultar o servidor DHCP.
- o link <https://www.cloudshark.org/captures/0009d5398f37> , você pode obter um exemplo de arquivo de captura com solicitações DHCP.

Start typing a Display Filter

✓ Apply

Clear

Filters ▾

💡 Analysis Tools ▾

📊 Graphs ▾

📄 Export ▾

⚙️ Profile

[illegible]



# Lendo Solicitações DHCP

- Em muitos casos, as opções na mensagem de descoberta do DHCP incluem o nome do host do cliente.
- Nosso objetivo será extrair os identificadores do cliente e do servidor dessa mensagem.



# Lendo Solicitações DHCP

```
from scapy.all import *
pcap_path = "packets_DHCP.cap"
packets = rdpcap(pcap_path)
for packet in packets:
    try:
        packet.show()
        options = packet[DHCP].options
        for option in options:
            if option[0] == 'client_id':
                client_id = option[1].decode()
            if option[0] == 'server_id':
                server_id = option[1]
            print('ServerID: {} | ClientID: {}'.format(server_id, client_id))
    except IndexError as error:
        print(error)
```

Note:

Lemos os pacotes DHCP do arquivo para extrair os identificadores de cliente e servidor de cada pacote.

```
from scapy.all import *
from collections import Counter
from prettytable import PrettyTable
packets = rdpcap('packets_DHCP.cap')
srcIP=[]
for packet in packets:
    if IP in packet:
        try:
            srcIP.append(packet[IP].src)
        except:
            pass
counter=Counter()
for ip in srcIP:
    counter[ip] += 1
table= PrettyTable(["IP", "Count"])
for ip, count in counter.most_common():
    table.add_row([ip, count])
print(table)
```

**Note:**

- Primeiro dizemos ao scapy para ler todos os pacotes do PCAP em uma lista, usando a função rdpcap.
- Pacotes no scapy possuem elementos; estamos lidando apenas com dados IP dos pacotes.
- Cada pacote possui atributos como IP de origem, IP de destino, porta de origem, porta de destino, bytes etc.

# Lendo Solicitações DHCP

- Ao executar o script anterior, vemos uma tabela com um resumo dos endereços IP e uma contagem para cada um:

```
$ sudo python read_pcap.py
```

```
+-----+-----+  
|      IP      | Count |  
+-----+-----+  
| 192.168.0.1  |    6  |  
| 192.168.0.3  |    4  |  
|   0.0.0.0    |    2  |  
+-----+-----+
```

# Escrevendo um Arquivo pcap

- Com o método **wrpcap()**, podemos armazenar os pacotes capturados em um arquivo pcap.
- No exemplo a seguir, capturaremos pacotes TCP para transmissões HTTP na porta 80 e salvaremos esses pacotes em um arquivo pcap.

Ao executar o script, capturaremos os primeiros 100 pacotes que possuem portas de destino 80 ou 443 para a interface de rede selecionada e os resultados serão armazenados no arquivo packets.pcap.

```
from scapy.all import *
def sniffPackets(packet):
    if packet.haslayer(IP):
        ip_layer = packet.getlayer(IP)
        packet_src=ip_layer.src
        packet_dst=ip_layer.dst
        print("[+] New Packet: {src} -> {dst}".format(src=packet_src,
                                                    dst=packet_dst))

if __name__ == '__main__':
    interfaces = get_if_list()
    print(interfaces)
    for interface in interfaces:
        print(interface)
        interface = input("Enter interface name to sniff: ")
        print("Sniffing interface " + interface)
        packets = sniff(iface=interface, filter="tcp and (port 443 or port
            80)", prn=sniffPackets, count=100)
    wrpcap('packets.pcap', packets)
```

► Dúvidas?

