



## **Universidade Do Minho**

Licenciatura em Engenharia de Telecomunicações e Informática

---

# **Criptografia e Segurança em Redes**

## **Trabalho Prático 1**

---

10 DE OUTUBRO DE 2023

Diogo Araújo a101778  
Fernando Mendes a101263  
Junlin Lu a101270

## Questão 1

- 1.1 Quais as diferenças observadas nos protocolos usados em cada acesso?
- 1.2 De que maneira as diferenças observadas podem afetar a proteção das propriedades de segurança do tráfego em rede?
- 1.3 Use a opção *Analyze → Follow* do Wireshark para reconstruir o fluxo de cada acesso. Descreva o resultado e discuta como as propriedades de segurança são afetadas por cada protocolo usado na camada de aplicação.

1.1 Através *Wireshark* conseguimos capturar os pacotes na rede de dois sites [www.scanme.org](http://www.scanme.org) e [www.owasp.org](http://www.owasp.org) como pretendido e por conseguinte ao analisar cada um deles conseguimos verificar que:

- O site *scanme* permite-nos visualizar o fluxo de diversos pacotes e como tal observar os protocolos HTTP, TCP e IP;
- O site *owasp* permite-nos visualizar o fluxo de diversos pacotes e com tal observar os protocolos HTTP, TCP, IP e HTTPS (TLS).

No.	Time	Source	Destination	Protocol	Length	Info
23	3.060138	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	74	62002 → 80 [FIN, ACK] Seq=1 Ack=1 Win=516 Len=0
24	3.060182	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	74	62003 → 80 [FIN, ACK] Seq=1 Ack=1 Win=516 Len=0
27	3.060441	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	86	62141 → 80 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
28	3.060568	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	86	62142 → 80 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
48	3.241700	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	TCP	86	80 → 62141 [SYN, ACK] Seq=0 Ack=1 Win=64800 Len=0 MSS=1392 SACK_PERM WS=128
49	3.241700	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	TCP	86	80 → 62142 [SYN, ACK] Seq=0 Ack=1 Win=64800 Len=0 MSS=1392 SACK_PERM WS=128
50	3.241700	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	TCP	74	80 → 62003 [ACK] Seq=1 Ack=2 Win=502 Len=0
51	3.241848	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	74	62141 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
52	3.241896	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	74	62142 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
53	3.241993	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	HTTP	688	GET / HTTP/1.1
54	3.242836	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	TCP	74	80 → 62002 [FIN, ACK] Seq=1 Ack=2 Win=507 Len=0
55	3.242858	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	74	62002 → 80 [ACK] Seq=2 Ack=2 Win=516 Len=0
56	3.432331	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	TCP	74	80 → 62141 [ACK] Seq=1 Ack=615 Win=64256 Len=0
57	3.432331	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	TCP	1466	80 → 62141 [ACK] Seq=1 Ack=615 Win=64256 Len=0 [TCP segment of a reassembled PDU]
58	3.432331	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	HTTP	1009	HTTP/1.1 200 OK (text/html)
59	3.432393	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	74	62141 → 80 [ACK] Seq=615 Ack=2328 Win=132096 Len=0
528	8.348571	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	TCP	74	80 → 62141 [FIN, ACK] Seq=2328 Ack=615 Win=64256 Len=0
529	8.348618	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	74	62141 → 80 [ACK] Seq=615 Ack=2329 Win=132096 Len=0
593	10.471122	2001:818:ea3c:3700:a573...	2600:3c01::f03c:91ff:fe...	TCP	74	62141 → 80 [FIN, ACK] Seq=615 Ack=2329 Win=132096 Len=0
629	10.702662	2600:3c01::f03c:91ff:fe...	2001:818:ea3c:3700:a573...	TCP	74	80 → 62141 [ACK] Seq=2329 Ack=616 Win=64256 Len=0

Figura 1: Análise do fluxo de pacotes *scanme*

No.	Time	Source	Destination	Protocol	Length	Info
2689	217.776092	2001:818:ea3c:3700:a573...	2606:4700:10::6816:1a4d	TCP	86	62242 → 443 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
2711	217.468068	2606:4700:10::6816:1a4d	2001:818:ea3c:3700:a573...	TCP	86	443 → 62242 [SYN, ACK] Seq=0 Ack=1 Win=65280 Len=0 MSS=1360 SACK_PERM WS=8192
2721	217.468468	2001:818:ea3c:3700:a573...	2606:4700:10::6816:1a4d	TCP	74	62242 → 443 [ACK] Seq=1 Ack=1 Win=131840 Len=0
2725	217.468946	2001:818:ea3c:3700:a573...	2606:4700:10::6816:1a4d	TLSv1.3	910	Client Hello
2735	217.501984	2606:4700:10::6816:1a4d	2001:818:ea3c:3700:a573...	TCP	74	443 → 62242 [ACK] Seq=1 Ack=837 Win=57344 Len=0
2739	217.505933	2606:4700:10::6816:1a4d	2001:818:ea3c:3700:a573...	TLSv1.3	361	Server Hello, Change Cipher Spec, Application Data
2748	217.537486	2001:818:ea3c:3700:a573...	2606:4700:10::6816:1a4d	TLSv1.3	138	Change Cipher Spec, Application Data
2751	217.538219	2001:818:ea3c:3700:a573...	2606:4700:10::6816:1a4d	TLSv1.3	166	Application Data
2752	217.538359	2001:818:ea3c:3700:a573...	2606:4700:10::6816:1a4d	TLSv1.3	568	Application Data
2756	217.587899	2606:4700:10::6816:1a4d	2001:818:ea3c:3700:a573...	TCP	74	443 → 62242 [ACK] Seq=288 Ack=1487 Win=65536 Len=0

Figura 2: Análise do fluxo de pacotes *wasp*

## 1.2

Dadas as análises feitas na alínea anterior conseguimos concluir que:

- O site *scanme* implementa o protocolo HTTP e como não tem qualquer encriptação associado é nos permitido analisar e ler a informação nos pacotes respetivos;
- O site *owasp* implementa o protocolo HTTPS que de uma forma prática e visível no *wireshark* se traduz pelo protocolo de segurança TLS o que por sua vez impossibilita a leitura do conteúdo intrínseca aos seus pacotes.

Desta forma conclui-se que o facto de haver um protocolo que se assegura da encriptação dos pacotes previne o fácil acesso à informação correspondente protegendo desta forma a comunicação e aumentando a sua segurança.

**1.3** O uso da função *Analyse : Follow : HTTP Stream / TCP Stream* do Wireshark, possibilita a reconstrução da comunicação constituída pelos pacotes correspondentes e desta forma analisar todo o seu conteúdo.

No caso do site *scanme* além de reconstruirmos a comunicação através do fluxo de pacotes conseguimos também ler todo o conteúdo do mesmo pois não tem qualquer encriptação que a proteja.

```
GET / HTTP/1.1
Host: www.scanme.org
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: _ga=GA1.2.160319370.1695813858; _gid=GA1.2.931436219.1696414934; _gat=1; _ga_ZLGVH7N8S=GS1.2.1696414935.2.1.1696415320.0.0.0

HTTP/1.1 200 OK
Date: Wed, 04 Oct 2023 10:28:57 GMT
Server: Apache/2.4.7 (Ubuntu)
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 2058
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
<head>
<title>Go ahead and ScanMe</title>
<meta name="viewport" content="width=device-width,initial-scale=1">
<meta name="theme-color" content="#2A0D45">
<link rel="preload" as="image" href="/images/site/logo.png" imagesizes="160px" imageset="/images/site/logo.png, /images/site/logo-2x.png 2x">
<link rel="preload" as="image" href="/shared/images/site/icons.svg">
<link rel="stylesheet" href="/shared/css/site.css?v=2">
<script async src="/shared/js/site.js?v=2"></script>
<link rel="stylesheet" href="/shared/css/site-foot.css?v=2" media="print" onload="this.media=all">
<link rel="stylesheet" href="/site.css">
<!-- Google Analytics Code -->
<script>
<link rel="preload" href="https://www.google-analytics.com/analytics.js" as="script">
<script>
(function(i,s,o,r,u,m){(i.GoogleAnalyticsObject)=r;(i[r]=i[r]||function(){(i[r].q=i[r].q||[]).push(arguments)};i[r].l=1*new Date;)(a=m.createElement(s)).async=!0;a.src=u;m.parentNode.insertBefore(a,m)})(window,document,"script","https://www.google-analytics.com/analytics.js","ga");
ga("create","UA-110094177-1","auto");
ga("send","pageview");
</script>
```

Figura 3: Reconstrução do fluxo no site *scanme*

No caso do site *owasp* ao reconstruirmos a comunicação através do fluxo de pacotes percebemos que não conseguimos ter acesso ao conteúdo intrínseco dessa comunicação devido ao protocolo HTTPS (TLS) responsável pela encriptação.



Figura 4: Reconstrução do fluxo no site *owasp*

Construa um *sniffer* baseado no Scapy e configure o filtro para capturar o tráfego correspondente às alíneas abaixo e que tenha como origem ou destino o sistema virtual (VM na Figura 1). Observe que deverá aplicar um filtro para cada alínea independentemente. Armazene as capturas em arquivos *pcap* para posterior análise.

2.1 Capture apenas pacotes ARP<sup>a</sup>;

2.2 Capture qualquer pacote TCP com um número de porta de destino 80. A partir da VM, volte a aceder as páginas da Questão 1 enquanto executa o programa no sistema *host*. Compare os resultados das capturas provenientes da Questão 1 e 2.

2.3 Extenda o filtro para capturar todo o tráfego com origem ou destino na subrede do sistema virtual. Qual a diferença no resultado da captura?

2.4 Discuta como o Scapy poderia ser usado para violar a propriedade de segurança da disponibilidade (*i.e.*, *availability*, discutida na aula teórica).

<sup>a</sup>Inclua no relatório capturas de imagem que demonstrem o resultado obtido.

## 2.1

```
from scapy.all import *
def print_pkt(pkt:Packet):
    pkt.show()

pkt = sniff(iface='Wi-Fi',filter='arp',prn = print_pkt)

wrpcap('D:\\3Ano\\3ano\\Cryptography\\TP1\\PCAP\\file_name.pcap', pkt)
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	HuaweiDe_80:db:8d	Broadcast	ARP	60	ARP Announcement for 0.0.0.0

Figura 5:Filtro de pacotes arp exemplo 1

```
1 from scapy.all import *
2 def print_pkt(pkt):
3     pkt.show()
4
5 pkt = sniff(iface="ens33", filter="arp", prn=print_pkt)
```

```
##### Ethernet #####
dst = ff:ff:ff:ff:ff:ff
src = 00:50:56:c0:00:08
type = ARP
##### ARP #####
hwtype = Ethernet (10Mb)
ptype = IPv4
hwlen = 6
plen = 4
op = who-has
hwsrc = 00:50:56:c0:00:08
psrc = 192.168.66.1
hwdst = 00:00:00:00:00:00
pdst = 192.168.66.2
##### Padding #####
load = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

##### Ethernet #####
dst = ff:ff:ff:ff:ff:ff
src = 00:50:56:c0:00:08
type = ARP
##### ARP #####
hwtype = Ethernet (10Mb)
ptype = IPv4
hwlen = 6
plen = 4
op = who-has
hwsrc = 00:50:56:c0:00:08
psrc = 192.168.66.1
hwdst = 00:00:00:00:00:00
pdst = 192.168.66.2
##### Padding #####
load = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

##### Ethernet #####
dst = ff:ff:ff:ff:ff:ff
src = 00:50:56:c0:00:08
type = ARP
##### ARP #####
hwtype = Ethernet (10Mb)
ptype = IPv4
hwlen = 6
plen = 4
op = who-has
hwsrc = 00:50:56:c0:00:08
psrc = 192.168.66.1
hwdst = 00:00:00:00:00:00
pdst = 192.168.66.2
##### Padding #####
load = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

Figura 6:Filtro de pacotes arp exemplo 2

## 2.2

Capturar pacotes TCP com um número de porta de destino 80. A diferença entre Q1 e Q2 é que estamos, especificamente em pacotes relacionados à porta destino 80(HTTP) no Q2.

Time	Source	Destination	Protocol	Length	Info
7037.98.884367	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	86	60752 → 80 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
7038.98.884367	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	86	60752 → 80 [SYN] Seq=0 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
7070.91.088534	2600:3c01::f03c:91ff:fe18:bb2f	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	TCP	86	80 → 60752 [SYN, ACK] Seq=0 Ack=1 Win=64800 Len=0 MSS=1440 WS=256 SACK_PERM
7074.91.088731	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	74	60752 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
7077.91.088770	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	74	60752 → 80 [ACK] Seq=1 Ack=1 Win=132096 Len=0
7078.91.088948	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	HTTP	688	GET / HTTP/1.1
7079.91.088954	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	68	[TCP Reset=151510] 60752 → 80 [RST] Seq=1 Ack=1 Win=132096 Len=0
7089.91.294159	2600:3c01::f03c:91ff:fe18:bb2f	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	TCP	74	80 → 60752 [ACK] Seq=1 Ack=615 Win=64256 Len=0
7090.91.294159	2600:3c01::f03c:91ff:fe18:bb2f	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	TCP	1466	80 → 60752 [ACK] Seq=1 Ack=615 Win=64256 Len=1392 [TCP segment of a reassembled PDU]
7091.91.294159	2600:3c01::f03c:91ff:fe18:bb2f	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	HTTP	1009	HTTP/1.1 200 OK (text/html)
7094.91.294332	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	74	60752 → 80 [ACK] Seq=615 Ack=2328 Win=132096 Len=0
7095.91.294341	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	74	[TCP Dup ACK 709481] 60752 → 80 [ACK] Seq=615 Ack=2328 Win=132096 Len=0
7170.92.543558	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	HTTP	696	GET / HTTP/1.1
7171.92.543572	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	68	[TCP Reset=151510] 60752 → 80 [RST] Seq=615 Ack=2328 Win=132096 Len=0
7173.92.727819	2600:3c01::f03c:91ff:fe18:bb2f	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	TCP	74	80 → 60752 [ACK] Seq=2328 Ack=1237 Win=64128 Len=0
7174.92.727819	2600:3c01::f03c:91ff:fe18:bb2f	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	TCP	1466	80 → 60752 [ACK] Seq=2328 Ack=1237 Win=64128 Len=1392 [TCP segment of a reassembled PDU]
7175.92.727819	2600:3c01::f03c:91ff:fe18:bb2f	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	HTTP	1000	HTTP/1.1 200 OK (text/html)
7176.92.727940	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	74	60752 → 80 [ACK] Seq=1237 Ack=4654 Win=132096 Len=0
7177.92.727956	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	74	[TCP Dup ACK 717671] 60752 → 80 [ACK] Seq=1237 Ack=4654 Win=132096 Len=0
7320.97.643357	2600:3c01::f03c:91ff:fe18:bb2f	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	TCP	74	80 → 60752 [FIN, ACK] Seq=4654 Ack=1237 Win=64128 Len=0
7330.97.643473	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	74	60752 → 80 [ACK] Seq=1237 Ack=4655 Win=132096 Len=0
7330.97.643473	2001:818:ea3c:3700:c8a7:9f5b:1e05:35f3	2600:3c01::f03c:91ff:fe18:bb2f	TCP	74	60752 → 80 [ACK] Seq=1237 Ack=4655 Win=132096 Len=0

Figura 7: Captura de pacotes porta destino 80 no scanme

tcpvnmcanme.pcap					
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help					
Apply a display filter ... <Ctrl-/>					
	Destination	Protocol	Length	Info	
1	d148:3807:7c7c	2600:3c01::f03c:91ff:fe18:bb2f	TCP	86	47764 → 80 [FIN, ACK] Seq=475 Ack=2329 Win=64128 Len=0 TS
2	d148:3807:7c7c	2600:3c01::f03c:91ff:fe18:bb2f	TCP	86	47752 → 80 [FIN, ACK] Seq=1 Ack=1 Win=64896 Len=0 TSval=3
3	d148:3807:7c7c	2600:3c01::f03c:91ff:fe18:bb2f	TCP	86	47752 → 80 [ACK] Seq=2 Ack=2 Win=64896 Len=0 TSval=321417
4	d148:3807:7c7c	2001:818:e4f1:1:5f88:1bb9	TCP	86	[TCP Keep-Alive] 58992 → 80 [ACK] Seq=423 Ack=889 Win=641
5	d148:3807:7c7c	2001:818:e4f1:1:5f88:1bb9	TCP	86	[TCP Keep-Alive] 58992 → 80 [ACK] Seq=423 Ack=889 Win=641
6	d148:3807:7c7c	2001:818:e4f1:1:5f88:1bb9	TCP	86	[TCP Keep-Alive] 58992 → 80 [ACK] Seq=423 Ack=889 Win=641
7	d148:3807:7c7c	2001:818:e4f1:1:5f88:1bb9	TCP	86	[TCP Keep-Alive] 58992 → 80 [ACK] Seq=423 Ack=889 Win=641
8	d148:3807:7c7c	2001:818:e4f1:1:5f88:1bb9	TCP	86	[TCP Keep-Alive] 58992 → 80 [ACK] Seq=423 Ack=889 Win=641
9	d148:3807:7c7c	2001:818:e4f1:1:5f88:1bb9	TCP	86	[TCP Keep-Alive] 58992 → 80 [ACK] Seq=423 Ack=889 Win=641
10	d148:3807:7c7c	2001:818:e4f1:1:5f88:1bb9	TCP	86	[TCP Keep-Alive] 58992 → 80 [ACK] Seq=423 Ack=889 Win=641

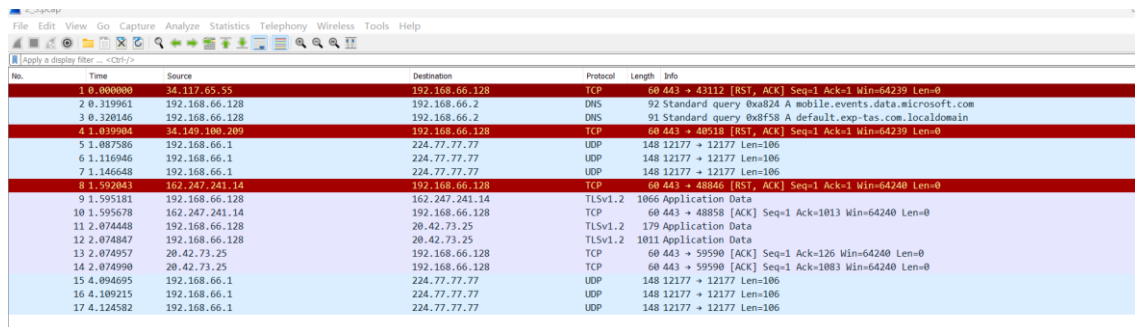
Figura 8: Captura de pacotes com porta destino 80 no wasp

**2.3** Através do filtro enunciado na linha 5 do código obtemos exatamente todo o fluxo na nossa sub-rede.

```
2.py x 2.2.py
2.3.py > ...
1 from scapy.all import *
2 def print_pkt(pkt):
3     pkt.show()
4
5 pkt = sniff(iface="ens33",filter="src net 192.168.1/24 mask 255.255.255.0", prn=print_pkt)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
| qname      = 'bam.nr-data.net.'
| qtype      = A
| qclass     = IN
an          = None
ns          = None
var         \
|###[ DNS OPT Resource Record ]###
| rname      = '.'
| type       = OPT
| rclass     = 1472
| extrcode   = 0
| version    = 0
| z          = 0
| rdlen      = 0
| rdata      \
|###[ Ethernet ]###
| dst        = 08:5b:56:f5:e8:25
| src        = 08:0c:29:44:43:f9
| type       = IPv4
|###[ IP ]###
| version    = 4
| ihl        = 5
| tos        = 0x0
| len        = 72
| id         = 20252
| flags      = 0
| frag       = 0
| ttl        = 64
| proto      = udp
| checksum   = 0x25b6
| src        = 192.168.66.128
| dst        = 192.168.66.2
| options    \
|###[ UDP ]###
| sport      = 42234
| dport      = domain
| len        = 52
| checksum   = 0x619
|###[ DNS ]###
| id         = 46555
| qr         = 0
| opcode     = QUERY
```

Figura 9: Programa para captura de pacotes com origem na nossa subrede



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	34.117.65.55	192.168.66.128	TCP	60	443 → 43112 [RST, ACK] Seq=1 Ack=1 Win=64239 Len=0
2	0.319961	192.168.66.128	192.168.66.2	DNS	92	Standard query 0xa824 A mobile.events.data.microsoft.com
3	0.320146	192.168.66.128	192.168.66.2	DNS	92	Standard query 0xbf58 A default.exp-tas.com.localdomain
4	1.035004	34.149.100.209	192.168.66.128	TCP	60	443 → 40510 [RST, ACK] Seq=1 Ack=1 Win=64239 Len=0
5	1.087586	192.168.66.1	224.77.77.77	UDP	148	12177 → 12177 Len=106
6	1.116946	192.168.66.1	224.77.77.77	UDP	148	12177 → 12177 Len=106
7	1.146648	192.168.66.1	224.77.77.77	UDP	148	12177 → 12177 Len=106
8	1.592043	162.247.241.14	192.168.66.128	TCP	60	443 → 48846 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
9	1.595181	192.168.66.128	162.247.241.14	TLSv1.2	1066	Application Data
10	1.595678	162.247.241.14	192.168.66.128	TCP	60	443 → 48858 [ACK] Seq=1 Ack=1013 Win=64240 Len=0
11	2.074448	192.168.66.128	20.42.73.25	TLSv1.2	179	Application Data
12	2.074847	192.168.66.128	20.42.73.25	TLSv1.2	1011	Application Data
13	2.074957	20.42.73.25	192.168.66.128	TCP	60	443 → 59590 [ACK] Seq=1 Ack=126 Win=64240 Len=0
14	2.074990	20.42.73.25	192.168.66.128	TCP	60	443 → 59590 [ACK] Seq=1 Ack=1083 Win=64240 Len=0
15	4.094695	192.168.66.1	224.77.77.77	UDP	148	12177 → 12177 Len=106
16	4.109215	192.168.66.1	224.77.77.77	UDP	148	12177 → 12177 Len=106
17	4.124582	192.168.66.1	224.77.77.77	UDP	148	12177 → 12177 Len=106

Figura 10: Captura de pacotes com origem na nossa subrede

*from scapy.all import \**: Isso importa todas as funções e classes da biblioteca Scapy, permitindo que use as funcionalidades dele.  
*def print\_pkt(pkt):* "print\_pkt" é definida, que aceita um argumento "pkt", que é um pacote de rede. Essa função será usada para mostrar informações sobre cada pacote capturado.  
Função *sniff()* do Scapy é usada para capturar pacotes  
*ens33* é uma interface de rede no VM. Define um filtro BPF (Berkeley Packet Filter) para capturar apenas pacotes TCP com uma porta de destino (destination port) igual a 80

**2.4** Dado que o scapy é utilizado com o intuito de praticar *Sniffing* de tráfego e através do mesmo conseguimos ter acesso ao fluxo de pacotes numa dada rede, permite-nos analisar e retirar informação que não deveriam ser privilegiadas a entidades que não aquelas que estão envolvidas na comunicação remetente a esse fluxo! Desta forma uma das violações que o *scapy* pode apresentar face à propriedade de segurança, disponibilidade provém de uma das ameaças que se denomina de *Information Disclosure*.

### Questão 3

Construa um programa capaz de fazer *spoofing* de pacotes ICMP. Configure o seu programa para enviar o **echo request** para o endereço IP 8.8.8.8. Configure o endereço de origem com o IP atribuído ao seu ambiente virtual. Execute o programa no sistema *host* tendo o Wireshark a capturar tráfego da *interface* ligada a mesma rede da VM.

**3.1** Analise a troca de pacotes correspondentes ao tráfego gerado. Discuta os resultados obtidos no teste<sup>a</sup>.

**3.2** Quais propriedades de segurança são violadas pelo programa criado nesta tarefa?

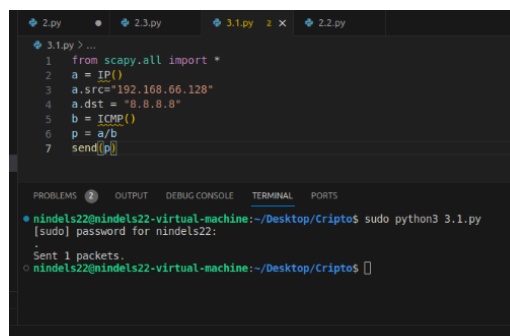
**3.3** Use um analisador de tráfego (Wireshark ou o seu programa criado na Tarefa 2) no ambiente virtual para observar se o pacote **echo reply** é entregue neste sistema<sup>a</sup>.

**3.4** Adapte o seu programa para que seja capaz de fazer *spoofing* de pacotes ICMP enviados pelo sistema virtual do seu ambiente de trabalho<sup>b</sup>.

<sup>a</sup>Inclua imagens no relatório que suportem a sua discussão.

<sup>b</sup>Inclua no relatório a descrição da adaptação feita ao programa original.

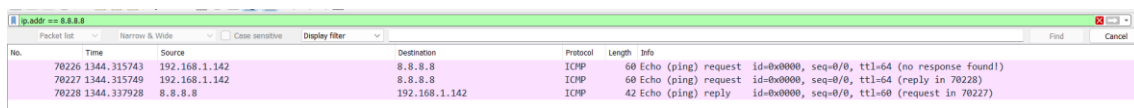
**3.1** Como o programa é capaz de fazer a troca de pacote ICMP, IP do VM como endereço de origem para os pacotes ICMP Echo Request. Isso é conhecido como "spoofing" e significa que os pacotes pareceriam ter vindo da sua VM, embora o IP real da origem fosse diferente.



```
3.1.py > ...
1 from scapy.all import *
2 a = IP()
3 a.src="192.168.0.128"
4 a.dst="8.8.8.8"
5 b = ICMP()
6 p = a/b
7 send(p)

nindels22@nindels22-virtual-machine:~/Desktop/Cripto$ sudo python3 3.1.py
[sudo] password for nindels22:
Sent 1 packets.
nindels22@nindels22-virtual-machine:~/Desktop/Cripto$
```

Figura 10: Programa para alterar a source



No.	Time	Source	Destination	Protocol	Length	Info
70226	1344.315743	192.168.1.142	8.8.8.8	ICMP	60	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found!)
70227	1344.315749	192.168.1.142	8.8.8.8	ICMP	60	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 70228)
70228	1344.337928	8.8.8.8	192.168.1.142	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=60 (request in 70227)

Figura 11: Captura de pacotes com ip origem 8.8.8.0

*from scapy.all import \**: Isso importa todas as funções e classes da biblioteca Scapy, permitindo que use as funcionalidades dele.

*a = IP(src=xxxxxx), a.dst = '8.8.8.8'* :define ip do origem e destino

*b = ICMP()*: é usado para criar um objetivo ICMP.

*P = a/b* : forma um pacote completo com um cabeçalho IP e um pacote ICMP encapsulado.

*send(p)*: enviar pacote.

### 3.2

As propriedades violadas são confidencialidade, Autenticidade, Integridade.



3.3

Sim, observamos echo reply no VM. IP 8.8.8.8 recebe estes pedidos, ele responde enviando pacotes ICMP Echo Reply de volta para a sua VM. Esses pacotes são a resposta às suas solicitações de ping.

ip.addr == 192.168.1.142					
No.	Time	Source	Destination	Protocol	Length Info
1	0.000000000	162.159.135.234	192.168.1.142	TLSv1.2	120 Application Data
2	0.000051056	192.168.1.142	162.159.135.234	TCP	68 44394 → 443 [ACK] Seq=1 Ack=53 Win=3031 Len=0 TSval=294
3	0.477610952	162.159.135.234	192.168.1.142	TLSv1.2	225 Application Data
4	0.477648452	192.168.1.142	162.159.135.234	TCP	68 44394 → 443 [ACK] Seq=1 Ack=210 Win=3030 Len=0 TSval=29
7	2.794173971	192.168.1.142	8.8.8.8	ICMP	44 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply
8	2.837730460	8.8.8.8	192.168.1.142	ICMP	62 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request

Figura 12: Captura de pacotes na VM

3.4

```
ex3.py > ...
1  from scapy.all import *
2
3  a = IP(src='192.168.1.142')
4  a.dst = '192.168.1.1'
5  b = ICMP()
6  p = a/b
7  send(p)
8
```

Figura 13: Programa para fazer spoofing de pacotes ICMP

410	10.724152	192.168.1.142	192.168.1.1	ICMP	60 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response found)
411	10.724158	192.168.1.142	192.168.1.1	ICMP	60 Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 412)
412	10.755111	192.168.1.1	192.168.1.142	ICMP	42 Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 411)
660	14.282887	162.159.135.234	192.168.1.142	TLSv1.2	596 Application Data

Figura 14: Captura de pacotes enviados pela VM